

**IBM**

**PowerPC 602  
Evaluation Board Kit  
User's Manual**

92G8620 000002

Second Edition (June 1997)

This edition of the *IBM PowerPC 602 Evaluation Kit User's Manual* applies to the IBM PowerPC 602 Evaluation Board Kit and to all subsequent versions of the 602 Evaluation Board Kit until otherwise indicated in new versions or technical newsletters.

**The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS MANUAL "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

IBM does not warrant that the contents of this publication or the accompanying source code examples, whether individually or as one or more groups, will meet your requirements or that the publication or the accompanying source code examples are error-free.

This publication could contain technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or program(s) described in this publication at any time.

It is possible that this publication may contain references to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country. Any reference to an IBM licensed program in this publication is not intended to state or imply that you can use only IBM's licensed program. You can use any functionally equivalent program instead.

No part of this publication may be reproduced or distributed in any form or by any means, or stored in a data base or retrieval system, without the written permission of IBM.

Requests for copies of this publication and for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

Address comments about this publication to:

IBM Corporation  
Department YM5A  
P.O. Box 12195  
Research Triangle Park, NC 27709

email: ppc400pubs@vnet.ibm.com

IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 199. All rights reserved.

Printed in the United States of America.

4 3 2 1

Notice to U.S. Government Users—Documentation Related to Restricted Rights –Use, duplication, or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corporation.

## **Patents and Trademarks**

IBM may have patents or pending patent applications covering the subject matter in this publication. The furnishing of this publication does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594, United States of America.

The following terms are trademarks of IBM Corporation:

PowerPC 602  
AIX  
AIXwindows  
IBM  
OS Open  
PowerPC  
PowerPC Architecture  
RISC System/6000  
RISCWatch  
RISCTrace

Other terms which are trademarks are the property of their respective owners.



# Contents

---

---

<b>About This Book .....</b>	<b>xix</b>
Who Should Use This Book.....	xix
How This Book is Organized .....	xx
Contacting the IBM Embedded Systems Solution Center.....	xxii
Related Publications.....	xxiii
<b>Overview of the 602 EVB .....</b>	<b>1-1</b>
Introducing the 602 EVB Hardware Components.....	1-1
602 Evaluation Board.....	1-1
Cables and Power Supply.....	1-1
Introducing the 602 EVB Software Support Package.....	1-2
ROM Monitor.....	1-2
RISCWatch Debugger .....	1-2
IBM High C/C++ Compiler .....	1-3
OS Open Real-Time Operating System.....	1-3
Dhrystone Benchmark Program.....	1-3
Application Tools.....	1-3
<b>Host System Requirements .....</b>	<b>2-1</b>
RS/6000 Host System Requirements.....	2-1
PC Host System Requirements.....	2-2
SUN Host System Requirements .....	2-3
<b>Installing the EVB Software .....</b>	<b>3-1</b>
RS/6000 Installation .....	3-1
EVB Software Support Package Installation - RS/6000.....	3-1
RISCWatch Debugger Installation - RS/6000 .....	3-4
PC Installation .....	3-4
EVB Software Support Package Installation - PC.....	3-4
RISCWatch Debugger Installation - PC .....	3-6
Sun Installation .....	3-6
EVB Software Support Package Installation - Sun .....	3-6
RISCWatch Debugger Installation - Sun.....	3-9
<b>Host Configuration.....</b>	<b>4-1</b>
RS/6000 Host Configuration.....	4-1
Serial Port Setup - RS/6000.....	4-1
Ethernet Setup - RS/6000.....	4-5
ROM Monitor-Debugger Communication Setup - RS/6000 .....	4-7
PC Host Configuration.....	4-7

Serial Port Setup.....	4-8
Ethernet Setup - PC .....	4-10
Windows 3.1 .....	4-11
Ethernet Setup - Windows 95.....	4-12
Ethernet Setup - Windows NT 3.51.....	4-13
ROM Monitor-Debugger Communication Setup - PC .....	4-13
Sun Host Configuration .....	4-13
Serial Port Setup - SUN .....	4-14
Ethernet Setup - SUN .....	4-14
ROM Monitor-Debugger Communication Setup - SUN.....	4-14
<b>602 EVB Connectors .....</b>	<b>5-1</b>
Serial Port Connectors .....	5-3
Ethernet Connectors.....	5-3
RISCWatch JTAG Debugger Connectors .....	5-6
Expansion/Test Interface Connector J1 .....	5-7
Setting the 602 EVB Jumpers .....	5-13
J4 Jumper Settings .....	5-13
J5 Jumper Settings .....	5-13
J6 Jumper Settings .....	5-13
J10 Jumper Settings .....	5-14
J17 through J20 Jumper Settings .....	5-14
Connecting the 602 EVB Hardware.....	5-15
Serial Port Connection .....	5-15
Ethernet Connection .....	5-16
Power Connector .....	5-18
Using a Terminal Emulator .....	5-19
RS/6000 Terminal Emulation .....	5-19
PC Terminal Emulation .....	5-20
Windows 3.1 and Windows NT Terminal Emulation.....	5-20
Windows 95 Terminal Emulation .....	5-20
SUN Terminal Emulation.....	5-21
Booting the PowerPC 602 on the EVB .....	5-22
<b>602 EVB Hardware.....</b>	<b>6-1</b>
PowerPC 602 Overview .....	6-1
Peripheral Components.....	6-1
Mechanical Specifications .....	6-4
Displays .....	6-4
Switches .....	6-5
Power Supply .....	6-5
Initialization.....	6-5
Ethernet Controller .....	6-6

602 EVB Memory Map .....	6-6
IRQ Controller.....	6-7
LCD Display.....	6-7
<b>602 EVB ROM Monitor .....</b>	<b>7-1</b>
ROM Monitor Source Code .....	7-1
Communications Features.....	7-2
Bootp and tftp Configuration to support ROM Monitor Loads.....	7-2
RS/6000 bootp and tftp configuration .....	7-2
PC bootp and tftp configuration .....	7-4
Automatic startup for Windows 3.1 and Windows NT 3.51 .....	7-5
Automatic startup for Windows 95.....	7-6
SUN bootp and tftp configuration.....	7-6
Accessing the ROM Monitor.....	7-8
ROM Monitor Operation .....	7-8
Monitor Selections and Submenus.....	7-10
Initial ROM Monitor Menu .....	7-11
Selecting Power-On Tests .....	7-12
Selecting Boot Devices.....	7-14
Changing IP Addresses .....	7-16
Using the Ping Test.....	7-18
Entering the Debugger.....	7-20
Disabling the Automatic Display .....	7-22
Displaying the Current Configuration .....	7-23
Saving the Current Configuration.....	7-24
Setting the Baud Rate for S1 Boots .....	7-25
S1 Boot .....	7-27
Exiting the Main Menu .....	7-29
ROM Monitor User Functions .....	7-31
Flash Update Utility .....	7-31
<b>602 EVB Sample Applications .....</b>	<b>8-1</b>
Overview.....	8-1
ROM Monitor Flash Image .....	8-2
Using the Software Samples .....	8-4
Building and Running the Dhrystone Benchmark .....	8-5
Building and Running the usr_samp Program .....	8-6
Building and Running the timesamp Program .....	8-7
Resolving Execution Problems.....	8-8
Using the Ping Test on the ROM Monitor to Verify Connectivity .....	8-8
bootp and tftp Servers (Daemons) for ROM Monitor loads.....	8-8
Using OS Open Functions.....	8-9

<b>Application Libraries and Tools.....</b>	<b>9-1</b>
OS Open Libraries.....	9-1
Using Libraries and Support Software.....	9-4
Serial Port Support Library.....	9-4
Boot Library(RAM) .....	9-4
Input/Output Support Library.....	9-4
PowerPC Low-Level Processor Access Support Library .....	9-5
ROM Monitor Ethernet IP Interface Library.....	9-5
Real-time Clock Interface Support Library .....	9-5
Integrated Ethernet IP Interface.....	9-5
Software Timer Tick Support Library.....	9-5
Device Drivers Supplied with the 602 EVB.....	9-6
Asynchronous Device Driver.....	9-6
Device Driver Installation.....	9-6
Device Installation .....	9-7
Opening Asynchronous Communication Ports.....	9-8
Reading and Writing .....	9-8
I/O Control .....	9-9
Polled Asynchronous I/O.....	9-11
Ethernet Device Driver .....	9-12
Device Driver Installation.....	9-12
Device Installation .....	9-12
Opening and Closing Ethernet Files.....	9-12
Reading and Writing .....	9-13
I/O Control .....	9-14
ENET_SET_CHANNEL.....	9-14
ENET_CLEAR_CHANNEL.....	9-14
ENET_QUERY_ADDRESS.....	9-14
Ethernet Device Driver Example .....	9-15
ROM Monitor Ethernet Device Driver.....	9-17
ROM Monitor Ethernet Installation and Initialization.....	9-17
Liquid Crystal Display Device Driver .....	9-17
LCD Device Driver Installation.....	9-17
LCD Device Installation .....	9-17
Opening LCD Device .....	9-18
Writing to LCD .....	9-18
I/O Control .....	9-18
Environment Bringup and Initialization .....	9-19
Board bootstrap.....	9-20
Environment Initialization .....	9-20
Tools.....	9-20
elf2rom .....	9-20

hbranch .....	9-23
eimgbld .....	9-25
split4.....	9-26
<b>602 EVB Function Reference .....</b>	<b>10-1</b>
Attributes and Threads .....	10-1
Async Safe Functions .....	10-1
Cancel Safe Functions.....	10-2
Interrupt Handler Safe Functions .....	10-2
602 EVB Functions.....	10-2
<b>Program Trace Calls .....</b>	<b>A-1</b>
Overview.....	A-1
MSGDATA Structure .....	A-1
Ptrace Definitions .....	A-4
RD_ATTACH (30) .....	A-5
Request data.....	A-5
Response data .....	A-5
RD_CONTINUE (7).....	A-6
Request data.....	A-6
Response data .....	A-6
RD_DETACH (31).....	A-7
Request data.....	A-7
Response data .....	A-7
RD_FILL (105) .....	A-8
Request data.....	A-8
Response data .....	A-8
RD_KILL (8) .....	A-9
Request data.....	A-9
Response data .....	A-9
RD_LDINFO (34) .....	A-10
Request data.....	A-10
Response data .....	A-10
RD_LOAD (101).....	A-12
Request data.....	A-12
Response data .....	A-12
RD_LOGIN (103) .....	A-13
Request data.....	A-13
Response data .....	A-13
RD_LOGOFF (104).....	A-14
Request data.....	A-14
Response data .....	A-14
RD_READ_D (2).....	A-15
Request data .....	A-15

Response data .....	A-15
RD_READ_FPR (12) .....	A-16
Request data .....	A-16
Response data .....	A-16
RD_READ_GPR (11).....	A-17
Request data .....	A-17
Response data .....	A-17
RD_READ_GPR_MULT(71).....	A-18
Request data .....	A-18
Response data .....	A-18
RD_READ_I (1).....	A-19
Request data .....	A-19
Response data .....	A-19
RD_READ_I_MULT (71).....	A-20
Request data .....	A-20
Response data .....	A-20
RD_READ_SPR (115) .....	A-21
Request data .....	A-21
Response data .....	A-21
RD_READ_SR (118).....	A-22
Request data .....	A-22
Response data .....	A-22
RD_STATUS (114) .....	A-23
Request data .....	A-23
Response data .....	A-23
RD_STOP_APPL (113).....	A-24
Request data .....	A-24
Response data .....	A-24
RD_WAIT (108).....	A-25
Request data .....	A-25
Response data .....	A-25
RD_WRITE_BLOCK (19).....	A-26
Request data .....	A-26
Response data .....	A-26
RD_WRITE_D (5) .....	A-27
Request data .....	A-27
Response data .....	A-27
RD_WRITE_FPR (15).....	A-28
Request data .....	A-28
Response data .....	A-28
RD_WRITE_GPR (14).....	A-29
Request data .....	A-29
Response data .....	A-29

RD_WRITE_I (4).....	A-30
Request data .....	A-30
Response data .....	A-30
RD_WRITE_SPR (112) .....	A-31
Request data .....	A-31
Response data .....	A-31
RD_WRITE_SR (119).....	A-32
Request data .....	A-32
Response data .....	A-32
RL_LDINFO (181).....	A-33
Request data .....	A-33
Response data .....	A-33
RL_LOAD_REQ(180) .....	A-34
Request data .....	A-34
Response data .....	A-34
<b>ROM Monitor Load Format.....</b>	<b>B-1</b>
Overview.....	B-1
Section Types.....	B-1
First Section .....	B-2
Text Section.....	B-3
Data Section .....	B-3
Symbol Section .....	B-3
Boot Header .....	B-3
<b>602 EVB Kit Bill of Materials .....</b>	<b>C-1</b>
Bill of Materials .....	C-1
<b>Index.....</b>	<b>X-1</b>



# Figures

---

---

Figure 5-1. 602 EVB Connectors .....	5-2
Figure 5-2. Nine-Pin Serial Port Connector.....	5-3
Figure 5-3. RJ-45 Ethernet Connector (Front View) .....	5-4
Figure 5-4. 15-Pin Receptacle .....	5-5
Figure 5-5. RISCWatch JTAG Header (Top View).....	5-6
Figure 5-6. 200 Pin Expansion/Test Interface Connector .....	5-7
Figure 5-7. Serial Port Connection.....	5-15
Figure 5-8. Wiring in a Crossover Cable.....	5-16
Figure 5-9. 10BaseT Ethernet Connection With Hub.....	5-17
Figure 5-10. Ethernet Connection Using Crossover Cable.....	5-17
Figure 5-11. Point-to-Point 10Base5/10Base2 Ethernet Connection.....	5-18
Figure 7-1. ROM Monitor Address Map .....	7-10
Figure 9-1. elf2rom Output File .....	9-22
Figure 9-2. Detail of patch file placement.....	9-24
Figure 9-3. hbranch Output Image .....	9-24



# Tables

---

---

Table 5-1. Serial Port Signal Assignments.....	5-3
Table 5-2. Ethernet Connector Pinout.....	5-4
Table 5-3. Ethernet Connector Descriptions .....	5-4
Table 5-4. 10Base5 Receptacle Signal Assignments .....	5-5
Table 5-5. RISCWatch JTAG Interface Connections and Resistors .....	5-6
Table 5-6. Connector J1 — Processor Expansion 1 .....	5-8
Table 5-7. J4 Jumper Description .....	5-13
Table 5-8. J5 Jumper Description .....	5-13
Table 5-9. J6 Jumper Description .....	5-13
Table 5-10. J10 Jumper Description .....	5-14
Table 5-11. J17 through J20 Jumper Descriptions .....	5-14
Table 5-12. J3 Power Supply Connections .....	5-19
Table 6-1. SIMM configurations .....	6-2
Table 6-2. Memory Access Cycles.....	6-3
Table 6-3. Status LED and LCD Displays .....	6-4
Table 6-4. 602 EVB Switches .....	6-5
Table 6-5. IRQ Interrupts and mask bits .....	6-7
Table 9-1. OS Open Libraries .....	9-1
Table 9-2. OS Open Libraries for the 602 EVB.....	9-4
Table 9-3. ioctl() Commands for Asynchronous Device Drivers .....	9-9
Table 9-4. ioctl() Commands for the LCD Device Driver.....	9-18
Table 10-1. Functions Specific to 602 EVB.....	10-2
Table A-1. RD_ATTACH Request Table.....	A-5
Table A-2. RD_ATTACH Response Table.....	A-5
Table A-3. RD_CONTINUE Request Table .....	A-6
Table A-4. RD_CONTINUE Response Table .....	A-6
Table A-5. RD_DETACH Request Table .....	A-7
Table A-6. RD_DETACH Response Table .....	A-7

Table A-7. RD_FILL Request Table.....	A-8
Table A-8. RD_FILL Response Table .....	A-8
Table A-9. RD_KILL Request Table.....	A-9
Table A-10. RD_KILL Response Table.....	A-9
Table A-11. RD_LDINFO Request Table.....	A-10
Table A-12. RD_LDINFO Response Table .....	A-10
Table A-13. RD_LOAD Request Table .....	A-12
Table A-14. RD_LOAD Response Table .....	A-12
Table A-15. RD_LOGIN Request Table.....	A-13
Table A-16. RD_LOGIN Response Table .....	A-13
Table A-17. RD_LOGOFF Request Table .....	A-14
Table A-18. RD_LOGOFF Response Table .....	A-14
Table A-19. RD_READ_D Request Table .....	A-15
Table A-20. RD_READ_D Response Table.....	A-15
Table A-21. RD_READ_FPR Request Table.....	A-16
Table A-22. RD_READ_FPR Response Table .....	A-16
Table A-23. RD_READ_GPR Request Table .....	A-17
Table A-24. RD_READ_GPR Response Table .....	A-17
Table A-25. RD_READ_GPR_MULT Request Table .....	A-18
Table A-26. RD_READ_GPR_MULT Response Table.....	A-18
Table A-27. RD_READ_I Request Table .....	A-19
Table A-28. RD_READ_I Response Table .....	A-19
Table A-29. RD_READ_I_MULT Request Table .....	A-20
Table A-30. RD_READ_I_MULT Response Table .....	A-20
Table A-31. RD_READ_SPR Request Table.....	A-21
Table A-32. RD_READ_SPR Response Table.....	A-21
Table A-33. RD_READ_SR Request Table .....	A-22
Table A-34. RD_READ_SR Response Table .....	A-22
Table A-35. RD_STATUS Request Table.....	A-23
Table A-36. RD_STATUS Response Table .....	A-23

Table A-37. RD_STOP_APPL Request Table .....	A-24
Table A-38. RD_STOP_APPL Response Table .....	A-24
Table A-39. RD_WAIT Request Table .....	A-25
Table A-40. RD_WAIT Response Table .....	A-25
Table A-41. RD_WRITE_BLOCK Request Table .....	A-26
Table A-42. RD_WRITE_BLOCK Response Table .....	A-26
Table A-43. RD_WRITE_D Request Table.....	A-27
Table A-44. RD_WRITE_D Response Table.....	A-27
Table A-45. RD_WRITE_FPR Request Table .....	A-28
Table A-46. RD_WRITE_FPR Response Table .....	A-28
Table A-47. RD_WRITE_GPR Request Table.....	A-29
Table A-48. RD_WRITE_GPR Response Table.....	A-29
Table A-49. RD_WRITE_I Request Table .....	A-30
Table A-50. RD_WRITE_I Response Table.....	A-30
Table A-51. RD_WRITE_SPR Request Table .....	A-31
Table A-52. RD_WRITE_SPR Response Table .....	A-31
Table A-53. RD_WRITE_SR Request Table .....	A-32
Table A-54. RD_WRITE_SR Response Table.....	A-32
Table A-55. RL_LDINFO Request Table .....	A-33
Table A-56. RL_LDINFO Response Table.....	A-33
Table A-57. RL_LOAD_REQ Request Table .....	A-34
Table A-58. RL_LOAD_REQ Response Table .....	A-34



# About This Book

---

This book contains the information you need to install and use the IBM® PowerPC™ 602™ Evaluation Board (EVB), a hardware and software development tool for the PowerPC 602 32-bit RISC microprocessor.

Connection of the 602 EVB to a host system is required for the exercises in this book. Supported host systems include:

- an IBM RISC System/6000™ workstation running AIX™ 3.2.5 (or higher)
- an IBM or compatible PC running one of the following
  - Windows 3.1 (or higher) and a TCP/IP package compliant with the Microsoft Windows Socket API definition
  - Windows 95
  - Windows NT 3.51
- a Sun SPARCstation 5, 10, or 20 workstation running Solaris 2.3 (or higher) or SunOS 4.1.3 (or higher)

The 602 EVB hardware module comes with a 602 controller, an Ethernet controller, 2MB flash memory, two serial ports, a one line LCD, 4 MB of EDO DRAM, a real-time clock calendar with 2KB NV RAM, and expansion and test interfaces. The reference design also includes technical specifications and schematics.

The 602 EVB software includes the ROM Monitor (resident in the flash memory on the board), ROM Monitor source code, IBM's OS Open real time operating system, sample application programs, application development libraries and tools, IBM's High C/C++ compiler, and IBM's RISCWatch, a source-level debugger that runs on the host.

## Who Should Use This Book

This book is for hardware and software developers who need to evaluate the 602 microprocessor and use the debugging features of the 602 EVB to support software development.

Users should understand hardware and software development tools, concepts, and environments. Specifically, users should understand:

- the host's operating system

- the PowerPC Architecture™ and implementation-specific characteristics of the PowerPC 602 microprocessor
- C and assembler language programming

## How This Book is Organized

This book contains the following chapters and appendixes:

- Chapter 1, “Overview of the 602 EVB,” describes the product, its hardware and software components, and its relationship with the software tools on the host.
- Chapter 2, “Host System Requirements,” lists the hardware and software requirements of the host system.
- Chapter 3, “Installing the EVB Software,” describes the software installation on the host system.
- Chapter 4, “Host Configuration,” describes the steps required to facilitate communications between the host computer and the 602 EVB.
- Chapter 5, “602 EVB Connectors ,” describes the EVB connectors and the procedures for connecting and configuring the 602 EVB hardware.
- Chapter 6, “602 EVB Hardware,” describes the hardware components and their functions in terms of the overall organization of the 602 EVB .
- Chapter 7, “602 EVB ROM Monitor,” describes the operations of the ROM monitor.
- Chapter 8, “602 EVB Sample Applications,” contains sample applications to be built, loaded onto the EVB, and run.
- Chapter 9, “Application Libraries and Tools,” describes the application libraries and host tools provided with the EVB software.
- Chapter 10, “602 EVB Function Reference,” lists the OS Open functions for the 602 EVB platform. The function calls are arranged alphabetically by function name.
- Appendix A, “Program Trace Calls,” describes the messages for interfacing a debugger on the host system to the ROM monitor on the 602 EVB.
- Appendix B, “ROM Monitor Load Format,” describes the load format requirements supported by the ROM monitor.
- Appendix C, “602 EVB Kit Bill of Materials,” contains a list of parts used on the 602 EVB.

## Conventions Used in This Book

This book follows the numeric and highlighting notation conventions based on those used in the RISC System/6000 and AIX publications.

## Numeric Conventions

In general, numbers are used exactly as shown. Unless noted otherwise, all numbers are in decimal, and, if entered as part of a command, are entered without format information.

In text, binary numbers are preceded by a “B” followed by the number enclosed in single quotes, for example:

B'010'

In commands, binary numbers are preceded by “0b” or “b” followed by the number, which may be enclosed in single quotes, for example:

0b010 or b'010'

In text, hexadecimal numbers are preceded by an “X” followed by the number enclosed in single quotes, for example:

X'1A7'

In commands, hexadecimal numbers are preceded by “0x” or “x” followed by the number, which may be enclosed in single quotes, for example:

0x1a7 or x'1a7'

In text, the hexadecimal digits A through F appear in uppercase. In commands, these digits are typically entered in lowercase.

## Highlighting Conventions

This book uses the following highlighting conventions:

- The names of invariant objects known to the software appear in bold type. In some text, however, such as in lists, no special typographic treatment is used. Examples of such objects include:
  - Function and macro names
  - Data types and structures
  - Constants and flags

Names of objects known to the software must be entered exactly as shown.

- Variable names supplied by user programs appear in italic type. In some text, however, such as in lists, no special typographic treatment is used. Examples of these objects include arguments and other parameters.
- No highlighting appears in code examples.

## Syntax Diagram Conventions

Throughout this book, diagrams illustrate the syntax for string formats and commands. The following list shows how to read these diagrams:

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.
- A ► symbol begins a diagram.
- A → symbol indicates continuation of a diagram on the next line.
- A ← symbol indicates continuation of a diagram from the previous line.
- A ◀ symbol terminates a diagram.
- Keywords are in regular type, and variables are in italics. Keywords must be typed exactly as shown.
- Keywords or variables on the main path of a diagram are required.

► keyword — *variable1* — *variable2* →

- Keywords or variables shown on branches below the main path are optional.

► keyword — [ *variable1* ] [ *variable2* ] →

- Keywords or variables can appear in a stack, indicating that only one item in a stack can be chosen. If an item in a stack is on the main path, you must choose an item from the stack. If all items in a stack are below the main path, you may choose an item from the stack.
- For example, in the following syntax diagram, you must choose either *variable1* or *variable2*. However, because *variable3* and *variable4* are below the main path, neither is required.

► KEYWORD — [ *variable1* ] [ *variable2* ] [ *variable3* ] [ *variable4* ] →

- A repeat separator is a returning arrow that surrounds a syntax element or group and shows that the element or group can be repeated.

► KEYWORD — [ *variable1* ] →

## Contacting the IBM Embedded Systems Solution Center

For information about the 602 EVB Kit and the IBM family of hardware and software products for embedded system developers, call the IBM Embedded Systems Solution Center at (919) 254-1810.

Please send any comments regarding this document to the following Internet address:

[ppc400pubs@vnet.ibm.com](mailto:ppc400pubs@vnet.ibm.com)

## Related Publications

Many of the following publications are included on the CD ROM that comes with the evaluation kit. The others are available from your IBM Microelectronics representative:

- **RISC System/6000 Publications**

*IBM RISC System/6000: POWERstation and POWERserver Hardware Technical Information General Architectures*, SA23-2643

- **AIX Publications**

This book refers to the following AIX publications. The words "IBM AIX Version 3.2 for RISC System/6000" are actually part of the title of each book; however, in all references to these books, those words are omitted.

*Assembler Language Reference*, SC23-2197

*Commands Reference*, Volume 1, SC23-2376

*Commands Reference*, Volume 2, SC23-2366

*Commands Reference*, Volume 3, SC23-2367

*Commands Reference*, Volume 4, SC23-2393

*Editing Concepts and Procedures*, GC23-2212

- **Embedded Application Binary Interface (EABI) Publications**

*PowerPC Embedded Application Binary Interface (EABI)*

*System V Application Binary Interface, Third Edition*, 0-13-0100439-5

*System V Application Binary Interface, PowerPC Processor Supplement*

- **IBM High C/C++ Publications**

The following list includes the books in the IBM High C/C++ library:

*IBM High C/C++ Programmer's Guide for PowerPC*, 92G6920

*IBM High C/C++ Language Reference for PowerPC*, 92G6923

*IBM ELF Assembler User's Guide for PowerPC*, 92G6921

*IBM ELF Linker User's Guide for PowerPC*, 92G6922

- **OS Open Publications**

The following list includes the books in the OS Open library:

*IBM OS Open Programmer's Reference*, Volume 1, 92G6911

*IBM OS Open Programmer's Reference*, Volume 2, 92G6912

*IBM OS Open User's Guide*, 92G6897

- **RISCWatch Debugger Publications**

*RISCWatch Debugger User's Guide*, 13H6964

- **PowerPC 6xx User's Manuals**

*PowerPC 602 RISC Microprocessor User's Manual*, MPR602UMU-01

*PowerPC 603 RISC Microprocessor User's Manual*, MPR603UMU-01

*PowerPC 603e RISC Microprocessor User's Manual*, MPR603EUM-01

*PowerPC 604 RISC Microprocessor User's Manual*, MPR604UMU-01

# Overview of the 602 EVB

---

---

This chapter introduces the hardware and software in the 602 EVB kit.

## 1.1 Introducing the 602 EVB Hardware Components

The 602 EVB kit contains the evaluation board with its power supply, line cord, serial port and Ethernet cables.

### 1.1.1 602 Evaluation Board

The 602 EVB is a full featured prototyping board which comes with the PowerPC 602 microprocessor, 2MB of flash memory (preprogrammed with the ROM Monitor), 4MB of DRAM, two serial ports, an Ethernet controller, a real-time clock, 2KB NV RAM, and an expansion interface connector. Four DRAM slots are provided to support up to 128MB when all four slots are populated with 32MB SIMMs.

The dual serial ports connect to a National NS16C552 serial communications controller. The Ethernet controller is a Fujitsu MB86964. The real-time clock/calendar is supplied by a SGS-Thomson MK48T02-120PC1.

Product documentation for devices other than the 602 can be obtained from the respective manufacturers. Configuration and addressing information for all these devices is included in the subsequent chapter on the 602 EVB hardware.

Header connectors are provided for optional test equipment such as the RISCWatch™ JTAG debugger. This tool allows non-intrusive hardware and software debug through the 602 EVB JTAG port. For more information on the RISCWatch JTAG tool, call the IBM Embedded Systems Solution Center at (919) 254-1810.

### 1.1.2 Cables and Power Supply

The 602 EVB kit includes a serial port interface cable for connecting the EVB serial port 1 to a terminal or terminal emulator running on the host.

**Note:** The Sun version of the EVB kit contains a male-to-male adapter to support connectivity between serial port 1 on the EVB and a serial port on the host.

An Ethernet crossover cable is provided in the kit to support direct Ethernet communication with the host system. A standard 10BaseT Ethernet connector is provided on the EVB. The Ethernet crossover cable is for direct connection to a single host and cannot be used with a hub or a building's Ethernet network.

A power supply with line cord is also provided with the 602 EVB kit.

## 1.2 Introducing the 602 EVB Software Support Package

The 602 EVB software support package consists of the ROM Monitor, ROM Monitor source code, the RISCWatch source level debugger for ROM Monitor and OS Open debug modes, the IBM OS Open real time operating system, several sample programs (including the Dhrystone benchmark program), and application development libraries and tools. The IBM High C/C++ compiler is also included.

### 1.2.1 ROM Monitor

The ROM Monitor program for the 602 EVB is supplied in the four 512KB flash memory modules on the board. This code initializes the 602 processor and the controllers for serial and Ethernet communications. By supporting communications with the host computer system, the ROM Monitor provides the means to load applications from the host onto the EVB and to debug them with the RISCWatch source level debugger.

The ROM Monitor is accessed through a terminal (or terminal emulator) attached to serial port 1 on the EVB. The RISCWatch debugger, when in ROM Monitor mode, runs on the host system, communicating with the ROM Monitor through serial port 2 or the Ethernet interface on the 602 EVB.

The ROM Monitor source code is provided primarily for customers interested in developing their own ROM versions. It is also provided so that debuggers other than RISCWatch may be integrated with the 602 EVB. Appendix A describes the trace calls that support communication between the RISCWatch debugger on the host and the ROM Monitor running on the 602 EVB.

### 1.2.2 RISCWatch Debugger

The RISCWatch source level debugger provides a window-based debugging environment for application programs running on the 602 EVB. The debugger can be used to load and execute application programs on the evaluation board. Debugger installation and usage for ROM Monitor and OS Open (non-JTAG) targets are addressed in the *RISCWatch Debugger User's Guide* included in the EVB kit. A sample debug session is included with the debugger.

### **1.2.3 IBM High C/C++ Compiler**

The IBM High C/C++ compiler is a globally optimizing compiler developed for the PowerPC family of processors. It produces executable code in Extended Link Format(ELF) file format. The version included in the software support package is a limited capacity version created specifically for the 602 EVB kit. It supports the compilation, assembly, and linkage of the sample application programs and the ROM Monitor source code. A full featured version of the IBM High C/C++ compiler is available from IBM. For more information call the PowerPC Embedded Systems Solutions Center at (919)254-1810.

### **1.2.4 OS Open Real-Time Operating System**

OS Open is a real-time operating system (RTOS) available for the PowerPC 400 Series and 60x processors. OS Open is designed to take full advantage of the power of the IBM PowerPC RISC processors. Also, because the OS Open environment is built in a scalable fashion, it can be configured to meet the functional requirements and memory constraints of a wide variety of embedded systems.

OS Open features:

- Hard real-time support, including deterministic execution, priority inheritance protocols, and priority ceiling protocols
- Board support packages for plug-and-play operation of popular board-level products
- Support for existing American National Standards Institute (ANSI) C and emerging POSIX standards
- Open network interfaces to support embedded systems in heterogeneous environments
- Scalable implementations to meet the requirements and constraints of a variety of embedded systems

The version of OS Open included in the EVB software contains a limited function kernel that limits the number of threads that can be in existence at one time. Additional details can be found in the readme file following software installation. A full function OS Open kernel is available from IBM. Contact the IBM Embedded Systems Solutions Center at (919)254-1810 for additional information.

### **1.2.5 Dhrystone Benchmark Program**

The Dhrystone benchmark is a commonly available integer benchmark. It is included as an example program to be built, loaded onto the evaluation board, and executed. The results of this benchmark may vary based on compiler options and the system environment in which it is run.

### **1.2.6 Application Tools**

Several host-based tools are provided to support ROM and application development on the 602 EVB.



# 2

## Host System Requirements

---

---

This chapter describes the hardware and software requirements of the host system to which the 602 EVB is to be connected. Supported host systems include:

- an IBM RS/6000 workstation running AIX 3.2.5 (or higher)
- an IBM or compatible PC running one of the following
  - Windows 3.1 (or higher) and a TCP/IP package compliant with the Microsoft Windows Socket API definition
  - Windows 95
  - Windows NT 3.51
- a Sun SPARCstation 5, 10, or 20 workstation running Solaris 2.3 (or higher) or SunOS 4.1.3 (or higher)

### 2.1 RS/6000 Host System Requirements

Hardware requirements of the host RS/6000 computer include:

- Approximately 25MB of free disk space. This space is required for the IBM High C/C++ compiler, the 602 EVB Software Support Package, and the RISCWatch debugger. When planning disk space usage, consider disk space requirements for AIX and any other software packages.
- Two available serial ports, one for terminal emulation and the other for host-to-EVB communications. Only one serial port is required if an Ethernet adapter is available for host-to-EVB communications. For better performance, an Ethernet connection is strongly recommended. Most RS/6000 computers come equipped with two serial ports and an Ethernet adapter. Please consult your RS/6000 literature for more details.
- A graphics display (IBM 6091 or similar), to display debugger screens

The following software must be installed on the host RS/6000 computer to run the debugger that communicates with the ROM Monitor on the 602 EVB:

- RISCWatch 3.3 or higher
- AIX Version 3.2.5 or higher
- AIX/Windows™ with X11R5 and Motif 1.2

AIX tools used to develop OS Open applications include:

- High C/C++ compiler for C programs
- asppc assembler for assembler and C language programs
- eimgbld, binary image build tool
- ELF linker/binder, to build OS Open applications for a target system
- split4, splits a ROM image into 4 separate files

IBM and other vendors provide numerous optional software development tools for AIX, including tools for:

- Computer-aided software engineering (CASE)
- Structured analysis and design
- Program understanding
- Code management and version control

## 2.2 PC Host System Requirements

Hardware requirements of the host PC include:

- IBM or compatible system unit. Minimum requirements: x486 DX2 50/66 MHz with 8 MB of RAM
- VGA/SVGA Display Monitor. Minimum required: VGA 640x480. Recommended: SVGA 1024x768
- Approximately 25MB of free disk space. This space is required for the IBM High C/C++ compiler, the 602 EVB Software Support Package, and the RISCWatch debugger. When planning disk space usage, consider disk space requirements for Windows and any other software packages.
- Two available serial ports, one for terminal emulation and the other for SLIP host-to-EVB communications. Since PC hardware varies greatly, you should consult your PC literature to determine the number of serial ports available. Only one serial port is required if an Ethernet adapter is available for host-to-EVB communications. For better performance, an Ethernet connection is strongly recommended. Establishing an Ethernet host-to-EVB connection will most likely require the installation of an Ethernet adapter card and some additional connectivity hardware since most PCs do not come equipped for Ethernet communications. That hardware might include any or all of the following:
  - For 10BaseT, an Ethernet 10BaseT network transceiver, a twisted pair cable, and a hub. At a minimum, a 10BaseT point-to-point connection will require the Ethernet crossover cable supplied with the EVB kit.

- For 10Base2, an Ethernet/IEEE 802.3 10Base5/10Base2 network transceiver, two BNC “T” type connectors, two terminating resistors, and a thin coaxial cable. At a minimum, a 10Base2 point-to-point connection will require one thin coaxial cable, an Ethernet/IEEE 802.3 10Base5/10Base2 network transceiver, two BNC “T” connectors, and two BNC terminating resistors.

The following software must be installed on the host PC to run the debugger that communicates with the ROM Monitor on the 602 EVB:

- RISCWatch 3.3 or higher
- Windows 3.1 or higher, Windows 95, or Windows NT 3.51

Windows 3.1 users require a TCP/IP package compliant with the Microsoft Windows Socket API definition. One such compatible TCP/IP package is Trumpet Winsock, a TCP/IP protocol stack available from the [www.trumpet.com](http://www.trumpet.com) Internet site. Windows 95 users who want to establish a SLIP host-to-EVB connection over a second serial port, require Trumpet Winsock as well, since the TCP/IP package that comes with Windows 95 does not support SLIP communications. Appropriate installation documentation can be found at the Trumpet site. Users should refer to the documentation for the terms and conditions of using Trumpet Winsock. Information regarding the setup and use of Trumpet Winsock can be found in the subsequent chapter on “Host Configuration”.

**Note:** Trumpet is not recommended for Windows 95 users already connected to a network since installing Trumpet may cause problems with previously defined networks. If the recommended Ethernet host-to-EVB connection is going to be used (instead of the SLIP host-to-EVB connection), Windows 95 users do **not** need to install Trumpet since the TCP/IP package that comes with Windows 95 can be used to establish the Ethernet connection.

## 2.3 SUN Host System Requirements

Hardware requirements of the host Sun workstation include:

- Approximately 25MB of free disk space. This space is required for the IBM High C/C++ compiler, the 602 EVB Software Support Package, and the RISCWatch debugger. When planning disk space usage, consider disk space requirements for the operating system and any other software packages.
- An available serial port for terminal emulation and an Ethernet (Attachment Unit Interface (AUI) or RJ-45) port for host-to-EVB communications. Most Sun SPARCstations come equipped with one serial port and an Ethernet (AUI) port. Consult your Sun literature for additional details.
- Any or all of the following hardware to establish an Ethernet connection between the EVB and the host:

- For 10BaseT, an Ethernet 10BaseT network transceiver, a twisted pair cable, and a hub. At a minimum, a 10BaseT point-to-point connection will require the Ethernet crossover cable supplied with the EVB kit.
- For 10Base2, an AUI (or thick Ethernet) adapter cable (or an AUI/Audio Adapter cable depending on your SPARCstation model and options - both are available from Sun), an Ethernet/IEEE 802.3 10Base5/10Base2 network transceiver, two BNC "T" type connectors, two terminating resistors, and a thin coaxial cable. At a minimum, a 10Base2 point-to-point connection will require one thin coaxial cable, an Ethernet/IEEE 802.3 10Base5/10Base2 network transceiver, two BNC "T" connectors, and two BNC terminating resistors.
- Consult your hardware documentation for additional information.

- A graphics display to display debugger screens

The following software must be installed on the Sun workstation to run the debugger that communicates with the ROM Monitor on the EVB:

- RISCWatch 3.3 or higher
- SunOS 4.1.3 (or higher) or Solaris 2.3 (or higher)
- OpenWindows 3.0 (SunOS 4.1.3) or 3.3 (Solaris 2.3)

# 3

## Installing the EVB Software

---

---

This chapter describes the procedures for installing the EVB software on the host system. Details of the software, its directories and their contents, are also given. Please refer to the section corresponding to your host system.

### 3.1 RS/6000 Installation

#### 3.1.1 EVB Software Support Package Installation - RS/6000

The software support package is installed from diskettes on an AIX host system using the system management interface tool (**smit**).

Before beginning the installation, you must have:

- **EVB for RS/6000** installation diskettes
- RISC System/6000, running AIX Version 3.2.5 or higher
- Superuser privileges on the AIX system

The method used to perform Steps 7 through 20 of the installation procedure depends on your version of **smit**. To select options, use the appropriate method for your version:

- In the X Window version, position the cursor and make selections using the mouse.
- In the character-based version, position the cursor using arrow keys and make selections using function keys.

The following procedure installs the EVB software support package:

1. Log in as **root** or use the AIX **su** command to become the superuser.
2. Use a **cd** command to change to the directory where the install image file will be stored.  
Typically, the directory **/usr/sys/inst.images** holds install image files. However, any directory can be used.
3. Insert the EVB installation diskette labeled “1 of *n*” (*n* may vary) into the diskette drive.
4. Run the following **restore** command to read the file **EVB.instal.Z** from the diskette into the working directory:

```
restore -f/dev/rfd0
```

5. Insert the rest of the EVB installation diskettes into the diskette drive when prompted.
6. After the diskettes are read, unpack the file:  
**uncompress EVB.instal.Z**
7. Run the following command to begin the installation via **smit**:  
**smit install\_latest**
8. Type the fully qualified path name of the file **EVB.instal** into the **Input device/directory for software** field.  
The path includes the directory selected in Step 2, for example,  
**/usr/sys/inst.images/EVB.instal**.
9. Press **Enter**.
10. Position the cursor on the **Software to install** line.
11. Select the **list** button (X Window version) or the **F4=List** function key (character-based version) to display a list of available software.
12. From the list, select the item or items appropriate for your platform and application.
  - To install the IBM High C/C++ Compiler, select the highc base item (ELF file format version only).
  - To install the complete OS Open distribution, select both the OS Open base and the OS Open platform specific items.
13. Select **OK** to complete the selection process and return to the **Install Software Products at Latest Available Level** window.
14. Ensure that the response for **Automatically install PREREQUISITE software** is “no”.  
For systems running AIX 4 or later, this field is called **AUTOMATICALLY install requisite software**.
15. Ensure that the response for **OVERWRITE existing version** is “yes”.  
For systems running AIX 4 or later, this field is called **OVERWRITE same or newer versions**.
16. Ensure that the response for **COMMIT Software** is “yes”.  
For systems running AIX 4 or later, this field is called **COMMIT software updates**.
17. Begin the installation by selecting **Do** or **OK**.
18. Select **OK** at the **ARE YOU SURE?** screen to continue the installation.
19. When the Command status is **OK**, file installation is complete.
20. Exit **smit**.

The IBM High C/C++ Compiler is installed in the **/usr/highcpc** directory tree and the EVB software support package in the **/usr/osopen** directory tree. It may be necessary to change ownership of these directories, their subdirectories and their contents if other users will require access to them. The **/usr/highcpc/bin** directory contains the files required for the IBM High C/C++ Compiler. Those files include:

- asppc - Assembler for assembler language programs
- ldppc - ELF linker/binder to build applications to be run on the EVB
- hcpc - High C/C++ compiler for C programs
- arppc - ELF library archiver

The **readme** file under the **/usr/highcpc** directory contains the latest information regarding the compiler and should be considered “must reading”.

If you installed the compiler into a directory other than **/usr/highcpc**, edit the **bin/hcpc.cnf** file, and locate the line near the top of the file that reads **HCDIR=/usr/highcpc**. Change this to reflect the directory that the compiler was installed into. Save your changes and exit the editor.

The **/usr/osopen** directory tree contains the files and tools that support OS Open application and ROM development. The **/usr/osopen** subdirectories and their contents are as follows:

- **/bin**

This directory contains several host based utilities used for application and ROM program development.

- elf2rom - creates a ROM image from an ELF executable file
- eimgbld - creates a ROM Monitor loadable image from an ELF executable file
- hbranch - places an absolute branch in the last address of a ROM image
- rambuild - creates an assembler source file that contains the files found in a specified directory
- split2 - splits a ROM image into two pieces
- split4 - splits a ROM image into four pieces
- tracefmt - post-processes OS Open trace snapshots for AIX 3.2.X
- trc41 - post-processes OS Open trace snapshots for AIX 4.1

- **/examples**

This directory contains many example OS Open programs.

- **/PLATFORM**

This directory contains the OS Open platform specific code for the platform included in your EVB kit. The directory is not literally named “PLATFORM”, but rather is named to identify the board and processor that was shipped with your kit. For example, if your platform was the 602 evaluation board, this directory might be named m602\_evb.

- README.TXT - contains the latest information regarding this release

- /include - contains OS Open include files
- /ld - contains dynamically loadable modules that can be run from OS Open's OpenShell
- /lib - contains OS Open libraries
- /m4 - contains assembler preprocessor include files
- /openbios - contains the source code for the ROM Monitor (detailed in a later chapter)
- /samples - contains samples programs that can be compiled and run

Considerable effort goes into providing a quality product with consistent documentation. To insure that our customers have the advantage of the latest software features and updated information, README.TXT may contain clarifications and/or additional information and should be considered "must reading".

- **/COMMENT.USR and COMMENT.DOC**

Please take the time to complete these user comment forms. Your feedback and suggestions will help us to improve our products and technical publications. Fax and email instructions are included in each of the files.

### **3.1.2 RISCWatch Debugger Installation - RS/6000**

Please refer to the *RISCWatch Debugger User's Guide* for debugger installation instructions. Be sure to follow the instructions for RS/6000 installation.

## **3.2 PC Installation**

### **3.2.1 EVB Software Support Package Installation - PC**

Before beginning the installation, you must have:

- **EVB for PC** installation diskettes
- PC running Windows 3.1 or higher, Windows 95, or Windows NT 3.51

The following procedure installs the EVB software support package:

**NOTE: For Windows NT users, we recommend that you logon as "root".**

1. Insert the installation diskette labeled "EVB - PC" and "1 of n" (*n* may vary) into diskette drive A:
2. Start Microsoft Windows if it is not active
3. Select Run... from the File pull-down of Program Manager or from the Start menu for Win95/NT
4. Type 'A:INSTALL' to run the installation program
5. Follow the installation program instructions

Once completed, the IBM High C/C++ Compiler is installed in the **\highcppc** directory tree and the EVB software support package in the **\osopen** directory tree. The **\highcppc\bin** directory contains the files required for the IBM High C/C++ Compiler. Those files include:

- asppc.exe - Assembler for assembler language programs
- ldppc.exe - ELF linker/binder to build applications to be run on the EVB
- hcpc.exe - High C/C++ compiler for C programs
- arppc.exe - ELF library archiver

The **readme** file under the **\highcppc** directory contains the latest information regarding the compiler and should be considered “must reading”.

The **\osopen** directory tree contains the files and tools that support OS Open application and ROM development. The **\osopen** subdirectories and their contents are as follows:

- **\bin**

This directory contains several host based utilities used for application and ROM program development.

- elf2rom.exe - creates a ROM image from an ELF file
- eimgbld.exe - creates a ROM Monitor loadable image from an ELF executable file
- hbranch.exe - places an absolute branch in the last address of a ROM image
- rambuild.exe - creates an assembler source file that contains the files found in a specified directory
- make.exe - supports the use of makefiles when building application programs
- bootpd.exe - bootp server to support ROM Monitor downlaods
- split2 - splits a ROM image into two pieces
- split4 - splits a ROM image into four pieces
- tftpd.exe - tftp server to support host-to-EVB file transfers

- **\examples**

This directory contains many example OS Open programs.

- **\PLATFORM**

This directory contains the OS Open platform specific code for the platform included in your EVB kit. The directory is not literally named “PLATFORM”, but rather is named to identify the board and processor that was shipped with your kit. For example, if your platform was the 602 evaluation board, this directory might be named m602\_evb.

- README.TXT - contains the latest information regarding this release
- \include - contains OS Open include files
- \ld - contains dynamically loadable modules that can be run from OS Open’s OpenShell
- \lib - contains OS Open libraries
- \m4 - contains assembler preprocessor include files
- \openbios - contains the source code for the ROM Monitor (detailed in a later chapter)
- \samples - contains samples programs that can be compiled and run

Considerable effort goes into providing a quality product with consistent documentation. To insure that our customers have the advantage of the latest software features and updated information, README.TXT may contain clarifications and/or additional information and should be considered "must reading".

- **\COMMENT.USER and \COMMENT.DOC**

Please take the time to complete these user comment forms. Your feedback and suggestions will help us to improve our products and technical publications. Fax and email instructions are included in each of the files.

### **3.2.2 RISCWatch Debugger Installation - PC**

Please refer to the *RISCWatch Debugger User's Guide* for debugger installation instructions. Be sure to follow the instructions for PC installation.

## **3.3 Sun Installation**

### **3.3.1 EVB Software Support Package Installation - Sun**

The software support package is installed from diskettes on a Sun host system using the **cpio** and **tar** commands.

Before beginning the installation, you must have:

- **EVB for Sun** installation diskettes
- a Sun SPARCstation 5, 10, or 20 workstation running SunOS 4.1.3 (or higher) or Solaris 2.3 (or higher)
- Superuser privileges on the Sun system

The procedures required for installing the EVB software support package vary depending on the operating system being used. Please follow the instructions corresponding to your operating system.

1. Log in as **root** or use the **su** command to become the superuser
2. Open at least two windows for this procedure
3. Use the **cd** command to change to the **/usr** directory
4. Insert the installation diskette labeled "EVB - Sun" and "1 of *n*" (*n* may vary) into the diskette drive.

### **Instructions for SunOS 4.1.3 (or higher) only:**

5. From the second window run the command:

```
cpio -ivB EVB_os4.tar.Z EVB.tar.Z EVB_hcppc.tar.Z < /dev/rfd0
```

where **/dev/rfd0** is the name of your diskette device.

6. When the system prompts you for a new volume, move to the first window and type **eject** to eject the diskette. Insert the next diskette.

7. Move to the second window and type the name of the diskette drive (**/dev/rfd0**) to continue the process.

8. If prompted for more diskettes, repeat the previous two steps. When finished, type **eject** to remove the final diskette.

9. Return to the first window and verify that the following files are installed under the **/usr** directory:

**EVB.tar.Z**

**EVB\_os4.tar.Z**

**EVB\_hcppc.tar.Z**

10. Run the following commands to unpack and install the files (**order is important**):

```
zcat EVB.tar.Z | tar xvf -
```

```
zcat EVB_os4.tar.Z | tar xvf -
```

```
zcat EVB_hcppc.tar.Z | tar xvf -
```

Installation for SunOS is complete. The tar.Z files may be removed to recover space.

### **Instructions for Solaris 2.3 (or higher) only:**

11. From the first window type **volcheck**. This creates a file called **/vol/dev/rdiskette0/unlabeled** (the diskette device name).

If the system pops up a message box saying the diskette format is unrecognized, ignore the message and cancel the message box. The name of the file created may be different on your system. You can use the **eject -q** command to see the actual name. The file name returned is the name that should be used in the subsequent steps.

12. From the second window run the command:

```
cpio -ivB EVB.tar.Z EVB_hcppc.tar.Z < /vol/dev/rdiskette0/unlabeled
```

where **/vol/dev/rdiskette0/unlabeled** is the name of your diskette device.

13. When the system prompts you for a new volume, move to the first window. Type **eject** if the system did not automatically eject the diskette. Insert the next diskette and type **volcheck**.
14. Move to the second window and type the name of the diskette drive (**/vol/dev/rdiskette0/unlabeled**) to continue the process.
15. If prompted for more diskettes, repeat the previous two steps. When finished, type **eject** to remove the final diskette.
16. Return to the first window and verify that the following files are installed under the **/usr** directory:  
**EVB.tar.Z**  
**EVB\_hcpc.tar.Z**

17. Run the following commands to unpack and install the files:

```
zcat EVB.tar.Z | tar xvf -  
zcat EVB_hcpc.tar.Z | tar xvf -
```

Installation for Solaris is complete. The tar.Z files may be removed to recover space.

The IBM High C/C++ Compiler is installed in the **/usr/highhcpc** directory tree and the EVB software support package in the **/usr/osopen** directory tree. It may be necessary to change ownership of these directories, their subdirectories and their contents if other users will require access to them. The **/usr/highhcpc/bin** directory contains the files required for the IBM High C/C++ Compiler. Those files include:

- **asppc** - Assembler for assembler language programs
- **ldppc** - ELF linker/binder to build applications to be run on the EVB
- **hcpc** - High C/C++ compiler for C programs
- **arppc** - ELF library archiver

The **readme** file under the **/usr/highhcpc** directory contains the latest information regarding the compiler and should be considered “must reading”.

If you installed the compiler into a directory other than **/usr/highhcpc**, edit the **bin/hcpc.cnf** file, and locate the line near the top of the file that reads **HCDIR=/usr/highhcpc**. Change this to reflect the directory that the compiler was installed into. Save your changes and exit the editor.

The **/usr/osopen** directory tree contains the files and tools that support OS Open application and ROM development. The **/usr/osopen** subdirectories and their contents are as follows:

- **/bin**

This directory contains several host based utilities used for application and ROM program

development.

- elf2rom - creates a ROM image from an ELF file
- eimgbl - creates a ROM Monitor loadable image from an ELF executable file
- hbranch - places an absolute branch in the last address of a ROM image
- rambuild - creates an assembler source file that contains the files found in a specified directory
- bootpd - bootp server to support ROM Monitor downlaods
- split4 - split ROM image into four sperate pieces

- **/examples**

This directory contains many example OS Open programs.

- **/PLATFORM**

This directory contains the OS Open platform specific code for the platform included in your EVB kit. The directory is not literally named “PLATFORM”, but rather is named to identify the board and processor that was shipped with your kit. For example, if your platform was the 602 evaluation board, this directory might be named m602\_evb.

- README.TXT - contains the latest information regarding this release
- /include - contains OS Open include files
- /ld - contains dynamically loadable modules that can be run from OS Open's OpenShell
- /lib - contains OS Open libraries
- /m4 - contains assembler preprocessor include files
- /openbios - contains the source code for the ROM Monitor (detailed in a later chapter)
- /samples - contains samples programs that can be compiled and run

Considerable effort goes into providing a quality product with consistent documentation. To insure that our customers have the advantage of the latest software features and updated information, README.TXT may contain clarifications and/or additional information and should be considered “must reading”.

- **/COMMENT.USER and /COMMENT.DOC**

Please take the time to complete these user comment forms. Your feedback and suggestions will help us to improve our products and technical publications. Fax and email instructions are included in each of the files.

### **3.3.2 RISCWatch Debugger Installation - Sun**

Please refer to the *RISCWatch Debugger User's Guide* for debugger installation instructions. Be sure to follow the instructions for Sun installation.



# 4

## Host Configuration

---

---

Several host configuration steps are required to facilitate communications between the host computer and the evaluation board. These steps are outlined in this chapter. Please refer to the section corresponding to your host system.

### 4.1 RS/6000 Host Configuration

RS/6000 configuration requires that you be the superuser of the host workstation. This is accomplished by logging in as **root** or by using the AIX **su** command to become the superuser.

#### 4.1.1 Serial Port Setup - RS/6000

The RS/6000 includes two serial ports to support communications via asynchronous data transfer. These ports are labeled S1 and S2 on the back of the RS/6000's system unit. When properly configured, one serial port can be used to connect a terminal emulator running on the host to the ROM Monitor running on the EVB, and the other to provide a **Serial Line Internet Protocol** (or **SLIP**) network interface between the host and the EVB to download applications. This section addresses the proper configuration of the S1 and S2 serial ports to support these connections. Details on setting up the terminal emulator are discussed in a later chapter. In this section, S1 and S2 refer to the respective serial ports on the host RS/6000, and SP1 and SP2 (labeled J27 and J25 on the board) to the respective serial ports on the EVB.

The connection of the terminal emulator running on the host to the ROM Monitor running on the EVB, is made through the S1 serial port on the RS/6000 and the SP1 serial port on the EVB. A connection between the S2 serial port on the host and the SP2 serial port on the EVB, provides a SLIP network interface to download application programs from the host to the EVB. If the recommended Ethernet connection is going to be used, the S2-to-SP2 SLIP connection is optional and does not need to be established.

Proper setup involves the configuration of **tty** devices for both the S1 and S2 serial ports on the host. **tty0** is used for the terminal emulator-to-ROM Monitor connection and **tty1** for the host-to-EVB SLIP connection. It is also necessary to establish a SLIP network interface between S2 on the host and SP2 on the EVB. The following steps should be taken to insure proper S1, S2 configuration:

1. Log in as **root** or the superuser (**su**)

2. Determine if the **tty0**, **tty1** devices already exist

- enter **smit**
- select **Devices**
- select **TTY**
- select **List All Defined TTYs**

Perform step 3 for each tty not listed.

Perform step 4 for each tty listed to insure that it is properly configured.

3. To add a **tty** device

- return to the **TTY** screen
- select **Add a TTY**
- select **tty rs232 Asynchronous Terminal**
- select **sa0** - Serial Port 1 (for ROM Monitor connection) when adding **tty0**  
OR **sa1** - Serial Port 2 (for EVB SLIP connection) when adding **tty1**
- select **s1** for the port number when adding **tty0**
- OR **s2** for the port number when adding **tty1**
- insure that the BAUD rate is **9600** when adding **tty0**  
OR that the BAUD rate is **38400** when adding **tty1**
- insure that the PARITY is **none**
- insure that the BITS per character is **8**
- insure that the Number of STOP BITS is **1**
- insure that Enable LOGIN is **disabled**

The default settings for all the other fields are satisfactory.

- select **Do** or hit **Enter**

Upon successful completion, a properly configured **tty** device is created and thus, step 4 can be skipped for the particular **tty** (**tty0** or **tty1**) added.

Remember to repeat this step, step 3, if both **tty0** and **tty1** needed to be added.

4. To properly configure a previously defined **tty** device

For systems running **AIX 3** :

- return to the **TTY** screen
- select **Change / Show Characteristics of a TTY**
- select **tty#** (where # = 0 or 1)
- select **Change / Show TTY Program**
- insure that the following fields are set to the indicated values:

TTY	tty#	(#=0 for tty0, 1 for tty1)
TTY type	tty	
TTY interface	rs232	
Description	Asynchronous Terminal	
Status	Available	
Location	00-00-S*-00	(*=1 for tty0, 2 for tty1)
Parent Adapter	sa#	(#=0 for tty0, 1 for tty1)
Port Number	s*	(*=1 for tty0, 2 for tty1)
Terminal Type	dumb	
Enable LOGIN	disable	

The other fields can remain at their default values.

- select **Do** or hit **Enter**
- upon successful completion, select **Done** or hit **PF3** to return to the **TTY** screen
- select **Change / Show Characteristics of a TTY**
- select **tty#** (where # = 0 or 1)
- select **Change/Show HARDWARE TTY Characteristics**
- insure that the BAUD rate is **9600** for **tty0**

OR that the BAUD rate is **38400** for **tty1**

- insure that the PARITY is **none**
- insure that the BITS per character is **8**
- insure that the Number of STOP BITS is **1**
- select **Do** or hit **Enter**

Upon successful completion, the **tty** device is properly configured.

For systems running **AIX 4** or later :

- return to the **TTY** screen
- select **Change / Show Characteristics of a TTY**
- select **tty#** (where # = 0 or 1)
- insure that the following fields are set to the indicated values:

TTY	tty#	(#=0 for tty0, 1 for tty1)
TTY type	tty	
TTY interface	rs232	
Description	Asynchronous Terminal	
Status	Available	
Location	00-00-S*-00	(*=1 for tty0, 2 for tty1)
Parent Adapter	sa#	(#=0 for tty0, 1 for tty1)
Port Number	s*	(*=1 for tty0, 2 for tty1)
Terminal Type	dumb	
Enable LOGIN	disable	

- insure that the BAUD rate is **9600** for **tty0**  
OR that the BAUD rate is **38400** for **tty1**
- insure that the PARITY is **none**
- insure that the BITS per character is **8**
- insure that the Number of STOP BITS is **1**  
The other fields can remain at their default values.
- select **Do** or hit **Enter**

Upon successful completion, the **tty** device is properly configured.

5. This last step establishes the SLIP network over the **tty1 device** between the host and the EVB. It's optional for those using the recommended Ethernet connection for host-to-EVB communications. This step is **not** required for **tty0** since it is being used simply for terminal emulation. Unlike a LAN interface, a SLIP connection is point to point. We first need to specify an IP address for the host and then an IP address for the other end of the SLIP connection, which in this case, is the evaluation board. To do this:

- enter **smit**
- select **Communication Applications and Services**
- select **TCP/IP**
- select **Further Configuration**
- select **Network Interfaces**
- select **Network Interface Selection**
- select **Add a Network Interface**
- select **Add a Serial Line INTERNET Network Interface**
- select **tty1**
- set the INTERNET ADDRESS field to the host IP address. An acceptable value would be **8.1.1.4**
- set the DESTINATION Address field to the evaluation board's IP address. An acceptable value would be **8.1.1.5**

Make a note of the addresses selected for the host and the evaluation board. They will be needed later.

- set the Network MASK to **255.255.240.0**
- insure that ACTIVATE is **yes**
- insure that the TTY PORT is **tty1**
- leave the BAUD RATE field blank
- leave the DIAL STRING field blank

- select **Do** or hit **Enter**

Upon successful completion, the SLIP Network Interface is established over **tty1** and the serial port setup is complete.

If this step fails, insure that a SLIP Network has not already been defined over **tty1**. To make this check, return to the **Network Interface Selection** screen in smit and select **List All Network Interfaces**. If **sl1** is listed then a network interface has already been defined for **tty1** and its characteristics may need to be changed. Return to the **Network Interface Selection** screen and select **Change>Show Characteristics of a Network Interface**. Select **sl1** and insure that the fields are set as stated previously in this step. (Note - there is no need to change the IP addresses in the INTERNET ADDRESS and DESTINATION Address fields if they have already been defined, but use of the above mentioned IP addresses is strongly recommended to maintain consistency with the rest of the documentation.) Make a note of the IP addresses chosen since they will be needed later during board setup.

#### **4.1.2 Ethernet Setup - RS/6000**

In addition to (or in place of) the SLIP connection, an Ethernet connection can be used for host-to-EVB communications. The Ethernet connection is made through an Ethernet adapter on the host and the 10BaseT connector on the EVB. Ethernet is much faster than SLIP and is recommended when downloading large applications on to the board or when using the RISCWatch debugger.

An Ethernet connection may require additional hardware. The 602 EVB supports connection via Standard Ethernet, twisted pair (10BaseT).

At a minimum, a 10BaseT connection requires a crossover Ethernet twisted pair cable (included in the kit) for point-to-point communications. If you want more than two nodes, you will need a hub and straight-through twisted pair cables.

Other hardware required will depend on the type of Ethernet adapter you have on your RS/6000 and whether the board is being connected to an existing Ethernet network. *AIX Communications Concepts and Procedures (GC23-2203, two volumes)* has additional information about the management and configuration of a TCP/IP network, including specifics as to how to configure an Ethernet network interface. Some of the basic steps are outlined below. You should consult your network administrator before attempting ethernet setup.

1. The host must be equipped to participate in a 10BaseT Ethernet network. This may require the installation of an Ethernet adapter card for your specific RS/6000 model and, as discussed previously, additional connectivity hardware. Consult the documentation included with the hardware for installation instructions. Most RS/6000 models come with Ethernet adapters already installed. They are labeled ET in the back of the RS/6000 system unit.
2. Assuming the host system is equipped with the appropriate Ethernet adapter, the Ethernet interface must be configured properly. To do this:

- log in as **root** or the superuser (**su**)
- enter **smit**
- select **Communication Applications and Services**
- select **TCP/IP**
- select **Further Configuration**
- select **Network Interfaces**
- select **Network Interface Selection**
- select **Add a Network Interface**
- select **Add a Standard Ethernet Network Interface**

Note - choose "**Standard Ethernet**" as opposed to "IEEE 802.3 Ethernet". If you receive an error message stating that there is "No available adapter", go to step 3 and skip the remaining items in this step, step 2.

- select **en0**
- set the INTERNET ADDRESS field to the host IP address. This value must be different from that used for the SLIP interface. It can be set to any convenient value if the Ethernet network is private for 602 EVB development purposes. An acceptable value would be **7.1.1.4**

Make a note of the IP address selected for the host system. It will be needed later. Note that an IP address for the evaluation board is not required as it was for the point-to-point SLIP network interface. An IP address for the EVB will, however, be required later on for the board setup.

- set the Network MASK field to **255.255.240.0**
- insure that ACTIVATE is **yes**
- insure that the Use Address Resolution Protocol is **yes**
- leave the BROADCAST ADDRESS blank
- select **Do** or hit **Enter**

Upon successful completion, a properly configured Ethernet interface has been added. The Ethernet setup is complete and step 3 need not be performed.

3. Perform this step only if you received the "No available adapter" error message when trying to **Add a Standard Ethernet Network Interface** in step 2. This message indicates that either the Ethernet adapter is missing (or possibly misplugged) or the Ethernet Network Interface already exists. To determine if the interface already exists:

- return to the **Network Interface Selection** screen in **smit**

- select **Change>Show Characteristics of a Network Interface**

If **en0** is **not** listed, insure that the RS/6000 host does have an Ethernet adapter and, if possible, that it is plugged correctly. If the adapter was misplugged, repeat step 2 to add the Ethernet Network Interface.

if **en0** is listed, then the Ethernet Network Interface already exists. Select **en0** and note the IP address listed for the INTERNET ADDRESS field. This value is the host's Ethernet IP address and will be needed later. If no IP address is listed, choose one. The IP address **7.1.1.4** can be used to maintain consistency with the menus and examples in this document. The Ethernet setup is complete.

#### **4.1.3 ROM Monitor-Debugger Communication Setup - RS/6000**

Before the RISCWatch Debugger can be used, some additional steps need to be taken to establish ROM Monitor-Debugger communications. These steps involve an update of the TCP/IP **services** file and a refresh of the TCP/IP **inetd** daemon.

To modify the **/etc/services** file, you need to log in as **root** or the superuser (**su**). The following lines must be added to the file:

```
osopen-dbg 20044/tcp  # for RISCWatch OS Open debug
osopen-dbg 20044/udp  # for RISCWatch rom_mon debug
```

The AIX **refresh -s inetd** command must then be run to inform the **inetd** daemon of the changes made to the **/etc/services** file.

## **4.2 PC Host Configuration**

As stated previously, PC users are required to have a TCP/IP package compliant with the Microsoft Windows Socket API definition. Unlike Windows 95 and Windows NT, Windows 3.1 does not include such a package. To determine if you will need to install a TCP/IP package on Windows 3.1, do the following:

- Select the **Main** icon from the Windows' Program Manager.
- Select the **File Manager** icon.
- Select **File** from the menu bar and choose **Search**.
- Perform a search for **winsock.dll** on your entire hard drive.

If the **winsock.dll** file exists, you probably have some compliant TCP/IP package already installed. **Workgroup for Windows** is a product that provides such a TCP/IP package. If the **winsock.dll** file does not exist, you need to install a TCP/IP package compliant with the Microsoft Windows Socket API definition. One such package, Trumpet Winsock, can be downloaded from the following Internet site: [www.trumpet.com](http://www.trumpet.com).

**Note:** Windows 95 users who want to establish a SLIP host-to-EVB connection over a second serial port, require Trumpet Winsock as well, since the TCP/IP package that comes with Windows 95 does not support SLIP communications. Trumpet is not recommended for Windows 95 users already connected to a network since installing Trumpet may cause problems with previously defined networks. If the recommended Ethernet host-to-EVB connection is going to be used (instead of the SLIP host-to-EVB connection), Windows 95 users do **not** need to install Trumpet since the TCP/IP package that comes with Windows 95 can be used to establish the Ethernet connection.

The following information is provided as a **guide** to installing the Trumpet Winsock code. It is not meant to be a replacement to the installation instructions contained at the Trumpet Internet site. It is provided to help clarify items which may be confusing.

1. Go to the Trumpet Software International's web site (<http://www.trumpet.com>) and find the installation information for Trumpet Winsock. You want to download the latest version which can be used for Windows 3.1 (must have 16 bit support). For example, version 3.0 (file twsk30c.exe) is a combined 16 bit/Windows 95 release. This version can be downloaded and used for an evaluation period of 30 days. Use beyond the evaluation period requires a purchase.
2. The downloaded version is usually a single file called a self extracting ZIP file (has an extension \*.exe). This file should be installed in a new directory (c:\trumpet, for example) and then executed. Execution is accomplished by going to the newly created directory and entering the name of the file. This will result in the creation of many more files in the new directory.
3. Read any '**README**' files carefully. Ethernet users are interested in directions concerning Packet Drivers because you will not be using a modem and you have already determined that a TCP/IP package does not exist on your system.
4. If the readme file does not direct you to do otherwise, execute 'install.exe' to start the installation process. You will be prompted for any required information. Note that you may be informed that a search will be done to rename any '**winsoc.dll**' files found. If you performed this check earlier, this file should not be found anywhere else on your hard drive.
5. If a 'setup' screen appears, you can defer entering any fields until a later time.
6. When installation is complete, reboot the system, and bring up Windows.

#### **4.2.1 Serial Port Setup**

Most PCs include two serial ports to support communications via asynchronous data transfer. These ports are sometimes referred to as communication or COM ports. These ports are usually accessed from the back of the system unit. This document refers to them as serial ports S1 and S2. You should consult your PC literature to determine how many serial ports are available on your unit and where they are located.

When properly configured, one serial port can be used to connect a terminal emulator running on the host to the ROM Monitor running on the EVB, and the other to provide a **Serial Line Internet Protocol** (or **SLIP**) network interface between the host and the EVB to download applications. The SLIP host-to-EVB connection is optional if the recommended Ethernet connection is going to be used for host-to-EVB communications. This section addresses the proper configuration of the S1 and S2 serial ports to support these connections. Users should also refer to the Windows on-line help for “Changing Serial Port Settings”.

The connection of the terminal emulator running on the host to the ROM Monitor running on the EVB, is made through the S1 serial port on the PC and the SP1 serial port on the EVB. The S1 port must be configured for a baud rate of 9600, 8 data bits, 1 stop bit, and no parity. The proper setting of these parameters is discussed later in the section on terminal emulation.

A connection between the S2 serial port on the host and the SP2 serial port on the EVB, provides a SLIP network interface to download application programs from the host to the EVB. This connection can be used in place of or along with the recommended Ethernet connection.

To establish a SLIP network over the S2 serial port for host-to-EVB communications, define a SLIP interface via the TCP/IP package being used. Since TCP/IP packages for PCs vary, users should consult their TCP/IP literature or their system administrator on how to establish the SLIP interface between the host and the EVB. The following IP addresses are suggested for the SLIP interface:

- PC host (source) : **8.1.1.4**
- Board (destination) : **8.1.1.5**

Make a note of the IP addresses selected since they will be needed later.

Trumpet Winsock users can use the following steps as a guide to establishing the SLIP interface:

1. Open the Trumpet Winsock by double clicking on the Trumpet Winsock icon in the Trumpet Winsock Files program group.
2. If setup was bypassed during installation, your connection should fail. A Trumpet Winsock window comes up indicating your connection status. Select **Setup** from the File menu to open the Setup dialog.
3. Set the **IP address** field to the IP address of the PC host: **8.1.1.4** is suggested to maintain consistency with this document.
4. Select **SLIP** under Drivers and then go to **Dialler settings**.
5. Select the appropriate **COMM port** (COM2 for example) to be used for SLIP communications.
6. Set the **Baud rate** to **38400**.

7. Disable **Hardware handshaking** and make sure **No automatic login** is selected. Use the default settings for the remaining options and/or check the help for more details.
8. Select **OK** from **Dialler Settings** and then **OK** from **Setup**.
9. Edit the **hosts** file found in the installed Trumpet directory to include both the PC host IP address and the board IP address. For example:
  - 8.1.1.4 local\_slip
  - 8.1.1.5 evb\_slip

After entering all the information, you may need to restart Trumpet Winsock for the network setup to take effect.

Prior to exiting Windows, we recommend terminating Trumpet Winsock (close the application). If you do not follow this recommendation, subsequent Trumpet starts may fail. If this occurs, you will need to reboot your system.

#### **4.2.2 Ethernet Setup - PC**

In addition to (or in place of) the SLIP connection, an Ethernet connection can be used for host-to-EVB communications. The Ethernet connection is made through an Ethernet adapter on the host and the 10BaseT connector on the EVB. Ethernet is much faster than SLIP and is recommended when downloading large applications on to the board or when using the RISCWatch debugger.

An Ethernet connection requires additional hardware. The 602 EVB supports connection via Standard Ethernet, twisted pair (10BaseT). This connection requires that the host PC be equipped with an appropriate Ethernet adapter. The host adapter is not included in the EVB kit. Please consult your PC and adapter documentation for requirements and installation instructions.

At a minimum, a 10BaseT connection requires a crossover Ethernet twisted pair cable (included in the kit) for point-to-point communications. If you want more than two nodes, you will need a hub and straight-through twisted pair cables.

Other hardware required will depend on the type of Ethernet adapter you have on your PC and whether the board is being connected to an existing Ethernet network. Please consult the documentation included with the adapter hardware for additional instructions.

Since TCP/IP packages for PCs vary, users should consult their TCP/IP documentation for information regarding the management and configuration of an Ethernet network interface. Establishment of an ethernet interface requires a host IP address. If the host PC is connected to an existing ethernet network, the host IP address should already be defined. Consult your network administrator on how to obtain the host's ethernet IP address and how to add the EVB to the existing network.

To maintain consistency with this document, the following IP addresses are suggested for the Ethernet interface :

- PC host (source) : **7.1.1.4**
- Board (destination) : **7.1.1.5**

Make a note of the IP addresses selected since they will be needed later.

#### **4.2.2.1 Windows 3.1**

Trumpet Winsock users can use the following steps as a **guide** to establishing a local Ethernet interface:

1. Trumpet Software International provides software which works with 'packet drivers'. When you first install your ethernet card, a set of different device drivers are provided. In order to use Trumpet Winsock, you will need to select a 'Packet Driver'. The Kingston ethernet card, provided with some RISCWatch packages, contains a packet driver that can be selected. If you buy an ethernet card that does not contain a packet driver, you can use the help option on the Trumpet menu bar to find out how you may be able to obtain a packet driver from the Internet. We will assume you have already followed the instructions for installing your ethernet card, have installed Trumpet Winsock, and have chosen a packet driver for use with Trumpet.
2. Read any '**README**' files carefully. Pay particular attention to any directions concerning Packet Drivers.
3. Follow the instructions for **Using the Trumpet Winsock over a packet driver** from the main Trumpet Help window. Follow the instructions for **Installing a packet driver and WINPKT**. At the time of this publication, the WINPKT program needed to be extracted from '<ftp://ftp.trumpet.com/winsock/winpkt.com>'. The ndis3pkt package, referred in the help as a replacement for winpkt, does not work unless you have WorkGroups for Windows, or some other windows package that runs NDIS.
4. Using the Trumpet help as a guide, your '**autoexec.bat**' file will need to have two lines added to get the ethernet communications working. The first line starts the packet driver you installed with your ethernet card. The proper name and syntax for this line should be identified in your ethernet card installation guide or in one of the files that came with the packet driver (i.e. the Kingston ethernet card has a '.doc' file that is part of the packet driver that describes how to invoke the driver). The second line to add is '**winpkt 0x60**' (vector 0x60 is usually the default vector to use).
5. After updating the 'autexec.bat' file, reboot the system to execute the changes.
6. From Windows, start Trumpet Winsock by double clicking on the Trumpet Winsock icon in the Trumpet Winsock Files program group.

7. If setup was bypassed during installation, your connection should fail. A Trumpet Winsock window comes up indicating your connection status. Select **Setup** from the File menu to open the Setup dialog.
8. Set the **IP address** field to the IP address of the PC host: **7.1.1.4** is suggested to maintain consistency with this document.
9. Select **Packet driver**, and set the **Vector** to **60**, **Netmask** to **255.255.240.0**, and **Gateway** to **0.0.0.0**.
10. Select **OK**.
11. Edit the **hosts** file found in the installed Trumpet directory to include both the PC host IP address and the board IP address. For example:
 

```
7.1.1.4 local_enet
7.1.1.5 evb_enet
```

After entering all the information, you may need to restart Trumpet Winsock for the network setup to take effect.

Prior to exiting Windows, we recommend terminating Trumpet Winsock (close the application). If you do not follow this recommendation, subsequent Trumpet starts may fail. If this occurs, you will need to reboot your system.

#### **4.2.2.2 Ethernet Setup - Windows 95**

A compliant TCP/IP package comes with Windows 95, so no TCP/IP package needs to be installed. If you haven't done so already, install the ethernet card on the host system according to the directions that came with the card.

To set the Host IP address for the ethernet connection:

- select the 'My Computer' icon from the desktop.
- select 'Control Panel'.
- select 'Network'.
- Add the appropriate "Adapter" network component for the ethernet adapter being used (if not already added).
- Add a "Protocol" network component of 'Microsoft - TCP/IP' (if not already added). Specify the IP address (**7.1.1.4** is recommended to maintain consistency with this document) and netmask (**255.255.240.0**) to be used.

**Note:** The "services" file that must be updated as part of the RISCWatch or evaluation kit installation is in directory "C:\WINDOWS".

The Evaluation Kit software was developed for Windows 3.1. Though it can be run successfully on Windows 95, certain restrictions apply. For example, file IDs need to be restricted to an eight character file name, and a three character file extension, or RISCWatch will not be able to locate source files.

#### **4.2.2.3 Ethernet Setup - Windows NT 3.51**

A compliant TCP/IP package comes with Windows NT, so no TCP/IP package needs to be installed. If you haven't done so already, install the ethernet card on the host system according to the directions that came with the card.

To configure TCP/IP for ethernet, double-click on the control panel icon followed by the network icon. Windows NT will prompt you through adding an ethernet adapter and TCP/IP. An IP address of **7.1.1.4** is recommended to maintain consistency with this document. A netmask of **255.255.240.0** should be used.

**Note:** The "services" file that must be updated as part of the RISCWatch or evaluation kit installation is in directory "C:\WINNT35\system32\drivers\etc".

The Evaluation Kit software was developed for Windows 3.1. Though it can be run successfully on Windows NT, certain restrictions apply. For example, file IDs need to be restricted to an eight character file name, and a three character file extension, or RISCWatch will not be able to locate source files.

#### **4.2.3 ROM Monitor-Debugger Communication Setup - PC**

Before the RISCWatch Debugger can be used, some additional steps need to be taken to establish ROM Monitor-Debugger communications. These steps involve an update of the TCP/IP **services** file and a restart of the TCP/IP package for the update to take effect.

Most PC TCP/IP packages place the **services** file under one of the TCP/IP package's subdirectories. Trumpet Winsock users should find the **services** file in the directory where the Trumpet files were installed. Windows 95 users should find the **services** file under "C:\WINDOWS\SERVICES". Windows NT users will find the **services** file under "C:\WINNT35\system32\drivers\etc". Users should consult their TCP/IP documentation or system administrator if they can not locate the file. The following lines must be added to the file:

```
osopen-dbg 20044/tcp # for RISCWatch OS Open debug  
osopen-dbg 20044/udp # for RISCWatch rom_mon debug
```

For the update to take effect, TCP/IP needs to be re-started. This may require a re-boot of the system and/or a restart of the TCP/IP package.

### **4.3 Sun Host Configuration**

Sun configuration requires that you be the superuser of the host workstation. This is accomplished by logging in as **root** or by using the **su** command to become the superuser.

### **4.3.1 Serial Port Setup - SUN**

The Sun workstation includes two serial ports to support communications via asynchronous data transfer. These ports are labeled Serial A and Serial B on the back of the Sun's system unit. Some SPARCstation models multiplex these two ports into one physical port labeled A/B (Use A if it's available since use of the B port requires a special de-multiplexing cable from Sun). This section refers to these ports as S1 and S2, respectively. When properly configured, one of the serial ports can be used to connect a terminal emulator running on the host to the ROM Monitor running on the EVB. This connection is made through the S1 serial port on the Sun and the SP1 serial port on the EVB.

The S1 port on the host must be configured for a baud rate of 9600, 8 data bits, 1 stop bit, and no parity. The proper setting of these parameters is discussed later in the section on terminal emulation.

### **4.3.2 Ethernet Setup - SUN**

Since all Sun SPARCstations come equipped with an ethernet (or AUI) port, an ethernet connection is used for host-to-EVB communications. The ethernet connection is made through the ethernet port on the host and the 10BaseT connector on the EVB.

An Ethernet connection requires additional hardware. The 602 EVB supports connection via Standard Ethernet, twisted pair (10BaseT).

At a minimum, a 10BaseT connection requires a crossover Ethernet twisted pair cable (included in the kit) for point-to-point communications. If you want more than two nodes, you will need a hub and straight-through twisted pair cables. Consult the documentation included with the hardware for additional information.

Establishment of an ethernet interface requires a host IP address. If the host SPARCstation is connected to an existing ethernet network, the host IP address should already be defined. Consult your network administrator on how to obtain the host's ethernet IP address and how to add the EVB to the existing network. Make a note of the host's IP address since it will be needed later.

If the host SPARCstation is not connected to an existing ethernet network, then a network between the EVB and the host must be established. The **ifconfig** command can be used to establish such a network. Users should consult their network administrator and Sun documentation for additional information. A host IP address of 7.1.1.4 is suggested to maintain consistency with this document. Make a note of the IP address selected since it will be needed later during board setup.

### **4.3.3 ROM Monitor-Debugger Communication Setup - SUN**

Before the RISCWatch Debugger can be used, the TCP/IP **services** file must be updated to allow ROM Monitor-Debugger communications.

To modify the **/etc/services** file, you need to log in as **root** or the superuser (**su**). The following lines must be added to the file:

```
osopen-dbg 20044/tcp  # for RISCWatch OS Open debug  
osopen-dbg 20044/udp  # for RISCWatch rom_mon debug
```



# 5

## 602 EVB Connectors

---

---

This chapter describes the 602 EVB connectors. The 602 EVB can be accessed through two serial ports, an Ethernet 10Base5 or 10BaseT connector, and the RISCWatch JTAG connection. A standard DIN connector is provided for power supply connection. A processor expansion interface is also provided.

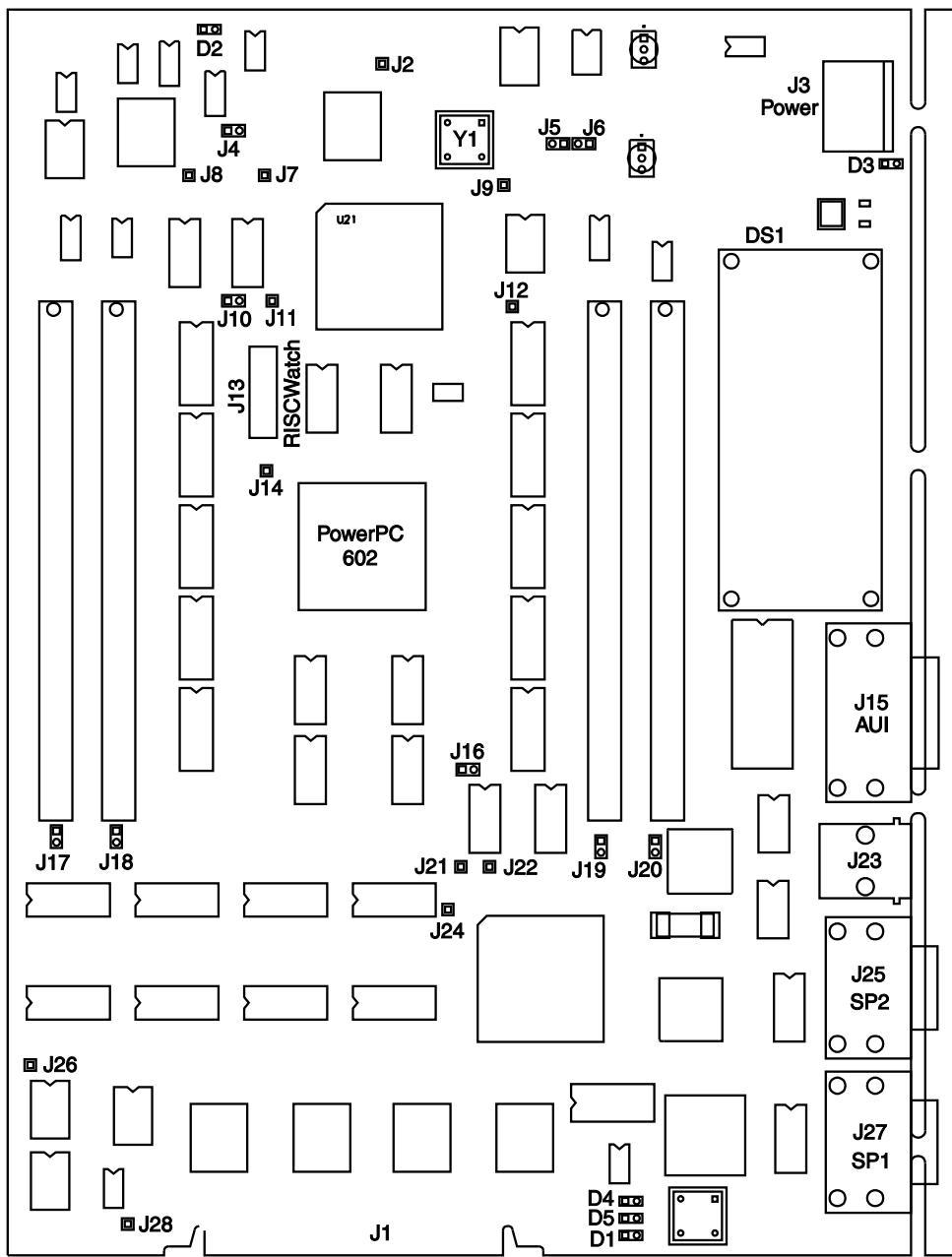
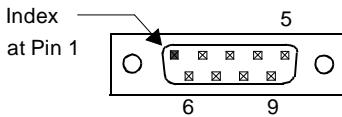


Figure 5-1. 602 EVB Connectors

## 5.1 Serial Port Connectors

Serial ports 1 (J27) and 2 (J25) are provided with standard nine-pin male right-angle connectors, as shown in Figure 5-3 below:



**Figure 5-2. Nine-Pin Serial Port Connector**

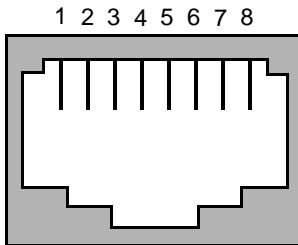
Table 5-1 describes the signal-to-pin assignments for serial ports 1 and 2:

**Table 5-1. Serial Port Signal Assignments**

Serial Port 1 (J27)		Serial Port 2 (J25)	
Pin Number	Signal Name	Pin Number	Signal Name
1	CD	1	CD
2	RXD	2	RXD
3	TXD	3	TXD
4	DTR	4	DTR
5	GND	5	GND
6	DSR	6	DSR
7	RTS	7	RTS
8	CTS	8	CTS
9		9	

## 5.2 Ethernet Connectors

The 602 EVB is provided with a standard 8-pin RJ-45 connector (J23) for 10BaseT and a 10Base5 connector (J15). The RJ-45 connector is shown in Figure 5-3.



**Figure 5-3. RJ-45 Ethernet Connector (Front View)**

Table 5-2 describes the 10BaseT Ethernet connector signals on the 602 EVB and the recommended mating connectors:

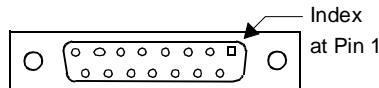
**Table 5-2. Ethernet Connector Pinout**

Pin Number	Signal Name	Description
1	TX+	Transmit Data +
2	TX-	Transmit Data -
3	RX+	Receive Data +
4,5	NC	No Connection
6	RX-	Receive Data -
7,8	NC	No Connection

**Table 5-3. Ethernet Connector Descriptions**

Connector Type	Receptacle Specifications	Mating Connector Specifications
10Base-T	RJ45 Right-Angle modular jack, AMP 555164-1	AMP 554739-1, 554169-1, or 5541170-1, Molex 90075-0130 or 90075-0132

Figure 5-4 shows the standard 15-pin right-angle 10Base5 receptacle (J15) on the EVB:



**Figure 5-4. 15-Pin Receptacle**

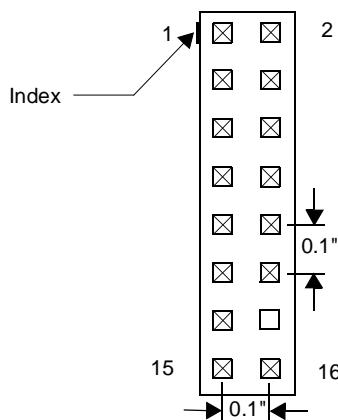
Table 5-4 describes the pin assignments of signals for the 10Base5 receptacle:

**Table 5-4. 10Base5 Receptacle Signal Assignments**

Pin Number	Signal Name	Pin Number	Signal Name
1	GND	9	$\overline{\text{CLSN}}$
2	CLSN	10	$\overline{\text{TRMT}}$
3	TRMT	11	GND
4	GND	12	$\overline{\text{RCV}}$
5	RCV	13	Power (+12V)
6	GND	14	GND
7	No Connect	15	No Connect
8	GND		

### 5.3 RISCWatch JTAG Debugger Connectors

The RISCWatch JTAG debugger connects to the 602 EVB JTAG port (J13) through a 2 × 8-pin header. This header is shown in Figure 5-5:



**Figure 5-5. RISCWatch JTAG Header (Top View)**

Placement of the RISCWatch JTAG header on the 602 EVB is indicated on the layout drawing in Figure 5-1, jumper J13. Signal names and positions on the headers are indicated in the following table:

**Table 5-5. RISCWatch JTAG Interface Connections and Resistors**

Header Pin #	I/O	Signal Name	Description	602 Pin1	Board Resistor <sup>2</sup>
1	Out	TDO	JTAG test data out	28	
2			To be left unconnected		
3	In	TDI	JTAG test data in	24	1KΩ PD
4		TRST		27	10KΩ PU
5			To be left unconnected		
6		+POWER <sup>3</sup>	Power (status signal, not processor V <sub>DD</sub> )		1KΩ SR <sup>4</sup>
7	In	TCK	JTAG test clock	26	10KΩ PU
8			To be left unconnected		
9	In	TMS	JTAG test mode select	25	10KΩ PU
10			To be left unconnected		
11	In	SRESET	Sreset_RISCWATCH (not Processor)	10	10KΩ PU
12			To be left unconnected		

**Table 5-5. RISCWatch JTAG Interface Connections and Resistors**

Header Pin #	I/O	Signal Name	Description	602 Pin <sup>1</sup>	Board Resistor <sup>2</sup>
13		HRESET	Signals a hardware reset	9	10KΩ PU
14		KEY	Pin in this position should be removed.		
15		CKSTP_OUT	Signals a checkstop	3	10K PU
16		GND	Ground		

<sup>1</sup>Pin numbers for PQFP packages

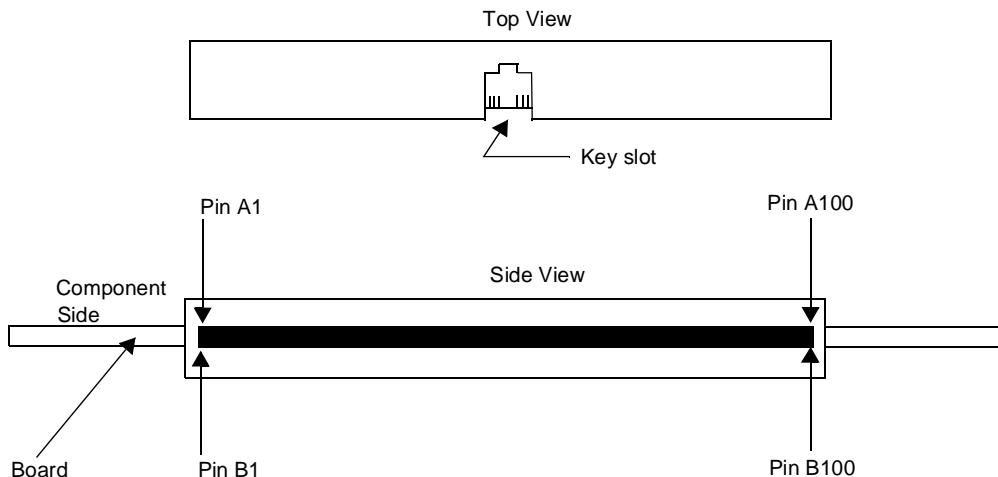
<sup>2</sup>PU = pullup resistor, SR = series resistor

<sup>3</sup>The +POWER signal is sourced from the target development board and is used as a reference signal. It should not be the power signal being supplied to the Processor (either +3.3V or +5V). It does not supply power to the RISCWatch hardware.

<sup>4</sup>This 1K ohm series resistor provides short current limiting protection only. If the resistor is present, it should be 1K ohm or less.

## 5.4 Expansion/Test Interface Connector J1

Connector J1 (processor expansion 1) , a MicroPax 200 pin connector, provides user access to all of the CPU control, address, and data signals. The expansion connector is designed to connect to a Berg MicroPax 200 pin connector. (P/N 91894-003 - female)



**Figure 5-6. 200 Pin Expansion/Test Interface Connector**

Table 5-6 describes the signal assignments for J1 processor expansion receptacle on the 602 EVB.

**Table 5-6. Connector J1 — Processor Expansion 1**

ROW A		ROW B	
Pin	Name	Pin	Name
1	ADD_LTCH0	1	RESET_EV_STATUS
2	ADD_LTCH1	2	BB_COM
3	ADD_LTCH2	3	BR_EV_STATUS
4	ADD_LTCH3	4	TS_COM
5	ADD_LTCH4	5	ARTRY_COM
6	ADD_LTCH5	6	VCC
7	ADD_LTCH6	7	OEBA_ADD_EV_STATUS
8	ADD_LTCH7	8	BUS_REQ_EXT
9	GND	9	OEAB_ADD_EV_STATUS
10	BG_EXT	10	GND
11	EN_SIMM2&3BITS_31_0	11	DAT_LTCH16
12	TA_EV_STATUS	12	DAT_LTCH17
13	GND	13	DAT_LTCH18
14	AACK_EV_STATUS	14	DAT_LTCH19
15	VCC	15	DAT_LTCH20
16	T32_EV_STATUS	16	DAT_LTCH21
17	ADD_LTCH8	17	DAT_LTCH22
18	ADD_LTCH9	18	DAT_LTCH23
19	ADD_LTCH10	19	SP1
20	ADD_LTCH11	20	GND

**Table 5-6. Connector J1 — Processor Expansion 1 (Continued)**

ROW A		ROW B	
Pin	Name	Pin	Name
21	ADD_LTCH12	21	SYSCLK4_DLY0
22	ADD_LTCH13	22	GND
23	ADD_LTCH14	23	Not Connected
24	ADD_LTCH15	24	Not Connected
25	GND	25	SP2
26	VCC	26	VCC
27	<u>MCP_EXT</u>	27	DATA_LTCH24
28	<u>TBEN_EXT</u>	28	DATA_LTCH25
29	<u>CKSTP_IN_EXT</u>	29	DATA_LTCH26
30	<u>SMI_EV_STATUS</u>	30	DATA_LTCH27
31	GND	31	DATA_LTCH28
32	ADD_LTCH16	32	DATA_LTCH29
33	ADD_LTCH17	33	DATA_LTCH30
34	ADD_LTCH18	34	DATA_LTCH31
35	ADD_LTCH19	35	<u>T32_EV_STATUS</u>
36	ADD_LTCH20	36	SP3
37	ADD_LTCH21	37	Not connected
38	ADD_LTCH22	38	GND
39	ADD_LTCH23	39	+12VOLTS_FUSED
40	GND	40	GND
41	SP6	41	SP4

**Table 5-6. Connector J1 — Processor Expansion 1 (Continued)**

ROW A		ROW B	
Pin	Name	Pin	Name
42	FPM_DRAM	42	DATA_LTCH32
43	EV_TENURE_TP	43	DATA_LTCH33
44	WAIT_TP	44	DATA_LTCH34
45	EXT_TENURE_TP	45	DATA_LTCH35
46	VAL_ADD_TP	46	DATA_LTCH36
47	LEAB_ADD_EV_STATUS	47	DATA_LTCH37
48	OEAB_DATA_EV_STATUS	48	DATA_LTCH38
49	OEBA_DATA_EV_STATUS	49	DATA_LTCH39
50	GND	50	GND
51	PARK	51	PFA16&D48
52	SP7	52	PFA17&D49
53	LEBA_ADD_EV_STATUS	53	TS1Z0&PFA18&D50
54	LEAB_DATA_EV_STATUS	54	TS1Z1&PFA19&D51
55	LEBA_DATA_EV_STATUS	55	TS1Z2&PFA20&D52
56	SP8	56	TRANSFER_BURST&D53
57	INTERLEAVE	57	TRANSFER_TYPE0&D54
58	VCC	58	TRANSFER_TYPE1&D55
59	GND	59	TRANSFER_TYPE2&D56
60	ADD_LTCH24	60	TRANSFER_TYPE3&D57
61	ADD_LTCH25	61	TRANSFER_TYPE4&D58
62	ADD_LTCH26	62	DATA_LTCH40

**Table 5-6. Connector J1 — Processor Expansion 1 (Continued)**

ROW A		ROW B	
Pin	Name	Pin	Name
63	ADD_LTCH27	63	DATA_LTCH41
64	ADD_LTCH28	64	DATA_LTCH42
65	ADD_LTCH29	65	DATA_LTCH43
66	ADD_LTCH30	66	DATA_LTCH44
67	ADD_LTCH31	67	DATA_LTCH45
68	GND	68	DATA_LTCH46
69	PFA0&D32	69	DATA_LTCH47
70	PFA1&D33	70	GLOBAL&59
71	PFA2&D34	71	CACHE_INHIBIT&D60
72	PFA3&D35	72	WRITE_THRU&D61
73	PFA4&D36	73	TRANSFER_CODE0&D62
74	PFA5&D37	74	TRANSFER_CODE1&D63
75	PFA6&D38	75	GND
76	PFA7&D39	76	VCC
77	DATA_LTCH0	77	DATA_LTCH48
78	DATA_LTCH1	78	DATA_LTCH49
79	DATA_LTCH2	79	DATA_LTCH50
80	DATA_LTCH3	80	DATA_LTCH51
81	DATA_LTCH4	81	DATA_LTCH52
82	DATA_LTCH5	82	DATA_LTCH53
83	DATA_LTCH6	83	DATA_LTCH54

**Table 5-6. Connector J1 — Processor Expansion 1 (Continued)**

ROW A		ROW B	
Pin	Name	Pin	Name
84	DATA_LTCH7	84	DATA_LTCH55
85	BE0&PFA8&D40	85	RESET_EXT
86	BE1&PFA9&D41	86	SP5
87	BE2&PFA10&D42	87	CS_EXT
88	BE3&PFA11&D43	88	GND
89	BE4&PFA12&D44	89	SERIAL_CLK_EXT
90	BE5&PFA13&D45	90	GND
91	BE6&PFA14&D46	91	IRQ_EXT
92	BE7&PFA15&D47	92	BG_EV_STATUS
93	DATA_LTCH8	93	DATA_LTCH56
94	DATA_LTCH9	94	DATA_LTCH57
95	DATA_LTCH10	95	DATA_LTCH58
96	DATA_LTCH11	96	DATA_LTCH59
97	DATA_LTCH12	97	DATA_LTCH60
98	DATA_LTCH13	98	DATA_LTCH61
99	DATA_LTCH14	99	DATA_LTCH62
100	Not Connected	100	DATA_LTCH63

## 5.5 Setting the 602 EVB Jumpers

The jumpers provided on the 602 EVB are detailed in the following tables:

### 5.5.1 J4 Jumper Settings

Memory bus width setting.

**Table 5-7. J4 Jumper Description**

J4	Description	Factory Setting
open	64 Bit Mode	
1-2	32 Bit Mode	←

### 5.5.1 J5 Jumper Settings

Memory page mode settings.

**Table 5-8. J5 Jumper Description**

J5	Description	Factory Setting
open	EDO DRAM	←
1-2	Fast Page Mode DRAM (not available at this time)	

### 5.5.1 J6 Jumper Settings

Memory Interleaved mode settings.

**Table 5-9. J6 Jumper Description**

J6	Description	Factory Setting
open	Interleaved Mode disabled	←
1-2	Interleaved Mode enabled	

### **5.5.1 J10 Jumper Settings**

Board clock frequency designation.

**Table 5-10. J10 Jumper Description**

J10	Description	Factory Setting
open	40 Mhz	←
1-2	33 Mhz	

**Note:** The oscillator placed in socket Y1 must match the J10 jumper setting.

### **5.5.1 J17 through J20 Jumper Settings**

J17 through J20 designate that the memory SIMM sockets that are populated.

**Note:** Read information about DRAM in Section 6.2 (4), pages 6-2, -3.

**Table 5-11. J17 through J20 Jumper Descriptions**

J17	Description	Factory Setting
open	SIMM0 not installed	
1-2	SIMM0 installed	←

J18	Description	Factory Setting
open	SIMM1 not installed	←
1-2	SIMM1 installed	

J19	Description	Factory Setting
open	SIMM2 not installed	←
1-2	SIMM2 installed	

J20	Description	Factory Setting
open	SIMM3 not installed	←
1-2	SIMM3 installed	

## 5.6 Connecting the 602 EVB Hardware

### 5.6.1 Serial Port Connection

To establish a working environment, the EVB must be connected to a host system. ROM Monitor access requires a connection between the J27 serial port on the board and the S1 (COM1) serial port on the host. Users must also establish a connection for debug and downloading applications from the host to the board. This connection is made over the SLIP or Ethernet network established during host configuration.

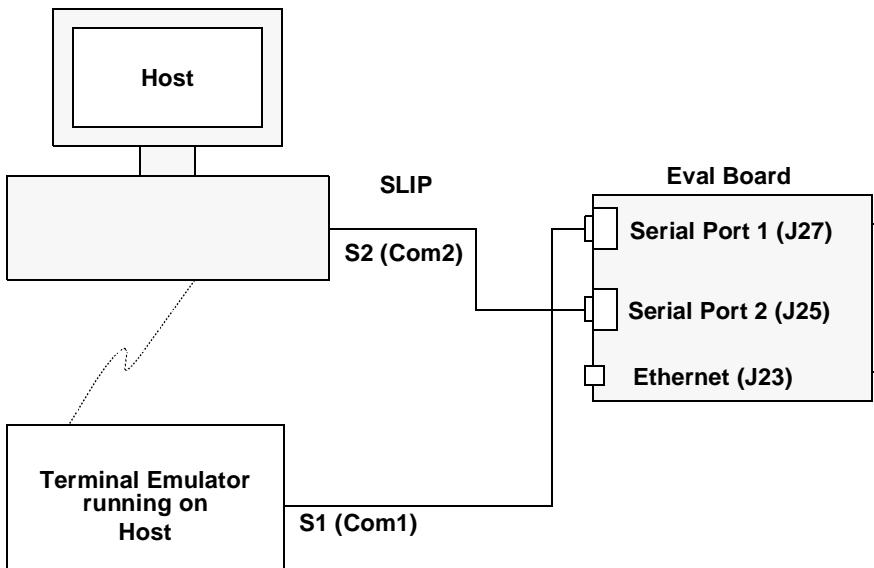


Figure 5-7. Serial Port Connection

Included in the 602 EVB kit is an interface cable supporting either 9-pin or 25-pin serial port connections. Use either the 9-pin or 25-pin, depending on the type of connector on the host. The cable is for connecting the J27 serial port 1 on the board to a terminal (or to a host running a terminal emulator). The board supports a second serial connection for communication over SLIP. This requires another interface cable (not provided) to attach to serial port 2 (J25).

Assuming a terminal emulator running on the host is going to be used for ROM Monitor access, connect the 9-pin serial port connector on one end of a cable to the J27 serial port on the EVB, and the other end of the same cable to the S1 (COM1) serial port on the host. The host end may require the 25-pin connector or a serial port adapter (not supplied) for connectivity. Sun SPARCstation users may require the 25 pin male-to-male adapter (included in the Sun 602 EVB kit) at the host end. If a SLIP connection is going to be used for host-to-EVB communications, connect a second cable (not provided) in a similar manner using J25 on the EVB and the S2 (COM2) serial port on the host.

### 5.6.2 Ethernet Connection

The Ethernet connection is made between the 10BaseT (J23) or 10Base5 (J15) connector on the 602 EVB and the Ethernet adapter on the host system. The board comes factory-set for 10BaseT communications and a 10BaseT crossover cable is included in the kit.

If the 10BaseT connection is to be used exclusively between the host and the EVB, the provided crossover cable can be used to directly connect the two nodes. Otherwise, a 10BaseT hub (not provided) must be used to connect the nodes together.

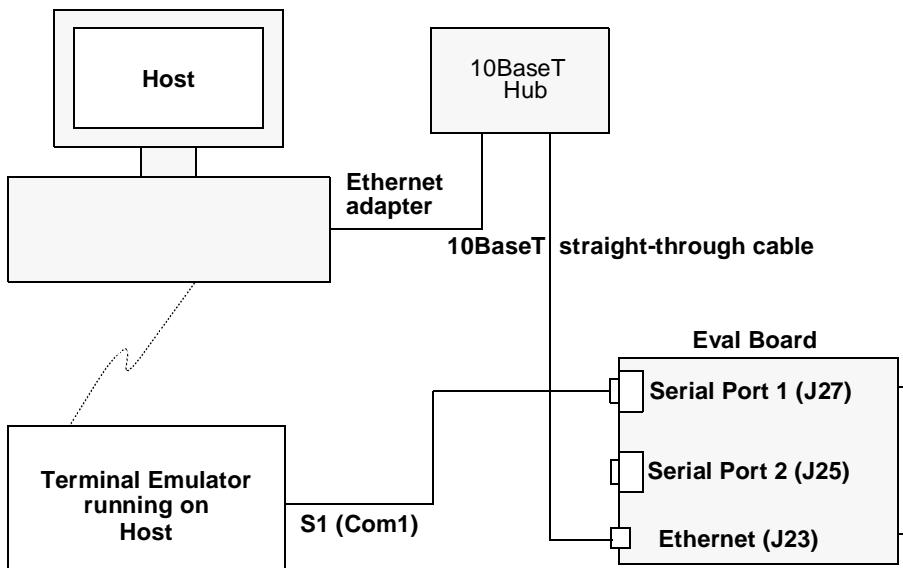
**Note:** The Ethernet 10BaseT crossover cable supplied will not work if plugged into a 10BaseT hub.

Figure 5-8 shows the connections and signal assignments required in a crossover cable:

RJ-45 Connector			10BaseT Cable		RJ-45
Twisted Pair	Signal Name	Pin		Pin	Signal Name
1	TD +	1		1	TD +
1	TD -	2		2	TD -
2	RD +	3		3	RD +
2	RD -	6		6	RD -
3,4	(Not used)	4, 5, 7, 8		4, 5, 7, 8	(Not used)

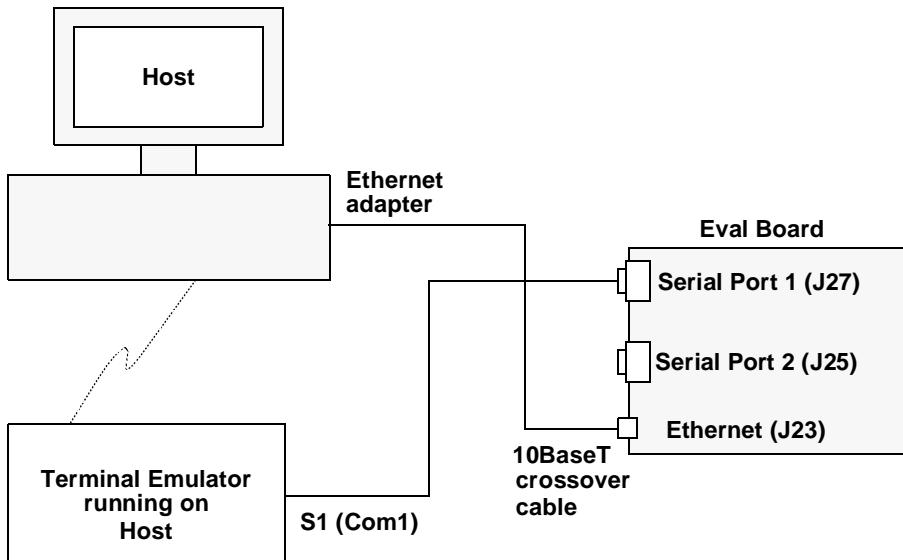
**Figure 5-8. Wiring in a Crossover Cable**

Figure 5-9 shows an Ethernet connection using a hub.



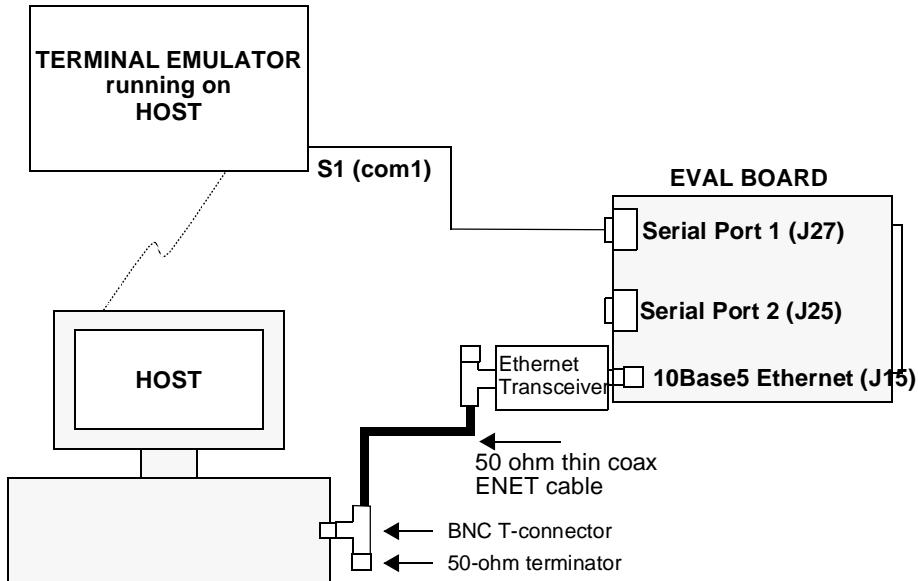
**Figure 5-9. 10BaseT Ethernet Connection With Hub**

Figure 5-10 shows a point-to-point Ethernet connection using the provided crossover cable.



**Figure 5-10. Ethernet Connection Using Crossover Cable**

If you wish to use **10Base5**, additional connectivity hardware (not included in the kit) is required. To connect from the EVB's 10Base5 connector to a 10Base2 (coax) network, additional hardware might include an Ethernet/IEEE 802.3 10Base5/10Base2 network transceiver, a 50 ohm thin coax ENET cable, two BNC "T" type connectors, and two 50 ohm terminating resistors. If the connection is going to be made to an existing Ethernet network, users should consult their Network Administrator to insure proper connectivity.



**Figure 5-11. Point-to-Point 10Base5/10Base2 Ethernet Connection**

**Note:** Both a SLIP and Ethernet connection can be used as long as both networks have been configured properly and the proper connections have been made.

### 5.6.3 Power Connector

The 602 EVB comes with a standard five-pin DIN connector for quick connect/disconnect to a power supply. A 1 Amp and a 4 Amp nanofuse are mounted on the board. A power supply and a line cord are provided with the 602 EVB. Power supply tolerances are  $\pm 5\%$  for the 5V and 12V supplies.

Table 5-12 defines the connections for the power supply connector:

**Table 5-12. J3 Power Supply Connections**

Pin Number	Function
1	GND
2	Frame Gnd
3	+5 V
4	No Connect
5	+12 V

**Warning:** use only the power supply provided in your EVB kit. It is recommended that the supply be connected to a 120V outlet prior to connection to the board.

## 5.7 Using a Terminal Emulator

The ROM Monitor transmits/receives data through serial port 1 (J27) on the evaluation board. Access to the ROM Monitor can be achieved by connecting a VT100 (or compatible) terminal directly to J27 on the EVB or by using a terminal emulator running on the host. When using a terminal emulator, access is obtained via a connection between J27 on the EVB and an available serial (or COM) port on the host system.

### 5.7.1 RS/6000 Terminal Emulation

The AIX Terminal Interface Program (TIP) can be used as a terminal emulator to support communications with the ROM Monitor. When properly configured, TIP connects the host RISC/6000 to a remote system, which in our case is the EVB. To set up TIP, do the following:

- log in as **root** or the superuser (**su**)
- go to the **/etc** directory (**cd /etc**)
- see if the file, **remote**, exists (**ls remote**). If the file does **not** exist, create it.
- using an editor, add the following line to the **remote** file (cut and pasters can find this line in the README.TXT file) :  
**tty0:dv=/dev/tty0:br#9600:el=^U^C^S^Q^D:ie=%\$:oe=^D:pa=none:**
- exit from **root**

TIP configuration is complete. Once all the host-to-EVB connections have been properly made and power has been supplied to the board, TIP can be activated by typing **tip tty0** at the AIX command prompt. After resetting the board, the ROM Monitor main menu should appear in the window where tip was activated. It may be necessary to hit the enter key once

or twice to get the menu to appear for the first time. Additional information on TIP can be found in *AIX Communications and Procedures (GC23-2203, two volumes)*.

Some useful escape sequences to know when using TIP include (Note - it may be necessary to hit the **Enter** key before entering these escape sequences.):

- ~? - help for TIP
- ~CTRL-D - instructs the TIP command to terminate the connection and exit
- ~# - sends a break to the remote system
- ~s script - starts recording of transmissions made by the remote system  
Recordings are made in the default **tip.record** file in the user's current directory
- ~s !script - stops recording of transmissions made by the remote system

**Note** - If a terminal emulator other than TIP is used, it must be configured for 9600 baud, eight bits per character, one stop bit, and no parity.

## 5.7.2 PC Terminal Emulation

### 5.7.2.1 Windows 3.1 and Windows NT Terminal Emulation

Once all the host-to-EVB connections have been properly made and power has been supplied to the board, the Windows Terminal program can be used as a terminal emulator to support communications with the ROM Monitor. To do this:

- from Windows Program Manager, select Accessories
- select Terminal
- select Settings
- select Communications
- select COM1 (or the appropriate COM port used for S1 serial port set-up)
- select Baud Rate **9600**, Data Bits **8**, Stop Bits **1**, Parity **None**
- select Flow Control **Xon/Xoff**
- select OK

After resetting the board, the ROM Monitor menu should appear in the Terminal window. If it does not, check for proper connectivity between the host and the board. If the ROM Monitor menu still does not appear, insure that the COM port has been properly enabled. This can be done by using the configuration utility on the host PC (see your PC documentation for more details).

### 5.7.2.2 Windows 95 Terminal Emulation

Once all the host-to-EVB connections have been properly made and power has been supplied to the board, the Windows 95 HyperTerminal program can be used as a terminal emulator to support communications with the ROM Monitor. The steps for setting up the terminal emulator connected to COM1 are as follows:

- select 'Start' from the Windows 95 task bar
- select 'Programs'
- select 'Accessories'
- select 'HyperTerminal'
- If you see a window that says "You need to install a modem before you can make a connection. Would you like to do this now?" click on "No", you do not need a modem for the evaluation board.
- select the 'Hypertrm' icon
- enter a name, for example "evb" and select an icon
- select the following:  
 Connect using Direct to Com 1(default)  
 Bits per second - **9600**  
 Data bits - **8** (default)  
 Parity - **None** (default)  
 Stop Bits - **1** (default)  
 Flow Control - **Xon/Xoff**
- select 'OK'

### 5.7.3 SUN Terminal Emulation

The Terminal Interface Program (TIP) can be used as a terminal emulator to support communications with the ROM Monitor. When properly configured, TIP connects the host Sun SPARCstation to a remote system, which in our case is the EVB. To set up TIP, do the following:

- log in as **root** or the superuser (**su**)
- go to the **/etc** directory (**cd /etc**)
- see if the file, **remote**, exists (**ls remote**). If the file does **not** exist, create it.
- using an editor, add the following line to the **remote** file (cut and pasters can find this line in the README.TXT file) :

```
tty0:dv=/dev/ttya:br#9600:el=^U^C^S^Q^D:ie=%$:oe=^D:pa=none:
```

- **exit** from root

TIP configuration is complete. Once all the host-to-EVB connections have been properly made and power has been supplied to the board, TIP can be activated by typing **tip tty0** at the command prompt. After resetting the board, the ROM Monitor main menu should appear in the window where tip was activated. It may be necessary to hit the enter key once or twice to get the menu to appear for the first time. If the ROM Monitor menu does not appear, consult your System Administrator - the **ttya** device may need to be modified. Additional information on TIP can be found in the online man pages by typing **man tip**.

Some useful escape sequences to know when using TIP include (Note - it may be necessary to hit the **Enter** key or **CTRL-D** before entering these escape sequences.):

- **~?** - help for TIP
- **~CTRL-D** - instructs the TIP command to terminate the connection and exit
- **~#** - sends a break to the remote system
- **~s script** - starts recording of transmissions made by the remote system  
Recordings are made in the default **tip.record** file in the user's current directory
- **~s !script** - stops recording of transmissions made by the remote system

**Note** - If a terminal emulator other than TIP is used, it must be configured for 9600 baud, eight bits per character, one stop bit, and no parity.

## 5.8 Booting the PowerPC 602 on the EVB

When the connectors have been installed and power is applied to the 602 EVB, pressing the Reset switch causes the 602 and the communications controllers to reset. After the ROM monitor initializes the 602 EVB, the monitor menu is displayed if a properly configured terminal (or terminal emulator) is attached to serial port 1 (J27) of the EVB. Details of ROM Monitor operation are provided in a later chapter.

# 6

## 602 EVB Hardware

---

---

Features of the 602 EVB include:

- 1) Memory
  - Contiguous addressing
  - DRAM, four SIMM sockets, up to 128MB
    - interleaved or non-interleaved accesses at 40 MHz bus speed
  - Four 512KB flash (2MB)
  - Wired for 32 bit operation
- 2) Battery-backed real-time clock
- 3) Two 16550-type serial ports
- 4) Ethernet
  - 10Base5 and 10BaseT
- 5) LCD - 1 line x 16 characters
- 6) Processor-dependent interface
  - For logic analyzer or for processor-bus peripheral

### 6.1 PowerPC 602 Overview

The PowerPC 602 is a low-power, low-cost implementation of the PowerPC microprocessor family of RISC microprocessors. For more information about the PowerPC 602 processor, refer to the *PowerPC 602 RISC Microprocessor User's Manual*.

### 6.2 Peripheral Components

- 1) Two Serial Ports

One National PC16552.

2) Clock/calendar and Non-volatile RAM

SGS Thompson MK48T02.

3) Ethernet Controller

Fujitsu MB86964.

This controller is used with 32Kx8 packet SRAM, National DP8392 coaxial transceiver interface, and both 10Base5 and 10BaseT connectors.

4) DRAM

The board is equipped with four 72-pin SIMM sockets, for 5 volt 60 nsec EDO DRAM. The DRAM control can be interleaved allowing memory accesses to be partitioned across multiple SIMMs. Board logic accepts up to 128MB of DRAM. SIMMs may be single-sided or double-sided. Refer to the following table for the proper SIMM installation. The 602 EVB comes with one 4Meg EDO SIMM.

**Table 6-1. SIMM configurations**

		SIMM0	SIMM1	SIMM2	SIMM3
32-Bit	Non-interleaved	All combinations are valid			
	Interleaved	X	X		
				X	X
		X	X	X	X
64-Bit	Non-interleaved	X		X	
			X		X
		X	X	X	X
	Interleaved	X	X	X	X

- All SIMM sockets must be identically populated (both single-sided or both double-sided, and both the same memory size).
- SIMM configuration is read by XILINX during power up. Reasserting RESET will not cause SIMM config to be read. If mismatch (D3) is lit, the configuration is wrong. Check for proper configuration options.

- Board jumpers specify the SIMMs installed. See Table 5-11 for jumper programming of SIMM size.

5) PPC602 Dram Controller

The DRAM controller for the 602 EVB is a FPGA that uses a Xilinx XC3190A-2. The PPC602 DRAM Controller supports 32- or 64-bit wide memory in both interleaved and not interleaved configurations at up to 40 MHz bus clock rate. Memory sizing is automatically determined by attempting to write and read back data of the SIMM modules, while physical presence requires the installation of jumpers J17 through J20. Memory width, 32 or 64 bits, and the option to interleave are further selected by jumpers J4 and J6. See "Setting the 602 EVB Jumpers" on page 5-13.

The following chart describes the memory access cycles needed for a cache line operation in terms of the bus clock, the memory width and whether or not interleave is enabled.

**Table 6-2. Memory Access Cycles**

	<b>Interleaved</b>	<b>Read</b>	<b>Write</b>	<b>Cycles</b>
32-Bit			X	5-2-2-2-2-2-2-2
		X		7-2-2-2-2-2-2-2
	X		X	5-1-1-1-1-1-1-1
	X	X		7-1-1-1-1-1-1-1
64-Bit			X	5-2-2-2
		X		7-2-2-2
	X		X	5-1-1-1
	X	X		7-1-1-1

For more information about the PPC602 DRAM Controller contact IBM support.

## 6) Flash

Four AM 29F040-120JC socketed devices.

There are four 512K byte flash chips on the 602 EVB. The ROM Monitor image is programmed across the four flash chips. Each flash chip holds a single byte from every word of the ROM Monitor image. For example, the first byte of each word is in the *byte 0* chip, the second byte of each word is in the *byte 1* chip, the third byte of each word is in the *byte 2* chip, and the fourth byte of each word is in the *byte 3* chip.

## 7) Interrupt Controllers

Xilinx FPGA.

## 8) Power-On-Reset

Dallas DS1233.

## 9) LCD Display

OptreX DMC16117(1 x 16) LCD display

## 6.3 Mechanical Specifications

The card outline is  $7\frac{7}{8}$  x  $10\frac{7}{8}$  inch. The card can be used as a stand-alone development card, lying horizontal on a bench. The card includes nylon standoffs to serve as "feet" for the card.

## 6.4 Displays

**Table 6-3. Status LED and LCD Displays**

Location	Type	Description
D1	LED	Ethernet collision detected (Red).
D2	LED	SIMM size mismatch (Red).
D3	LED	Power good indicator (Green)
D4	LED	Ethernet transmit indicator (Yellow).
D5	LED	Ethernet Link indicator (Green).
DS1	LCD	OptreX DMC16117(1 x 16) LCD display.

## 6.5 Switches

There are two switches on the 602 EVB. The Reset switch (U2) on the board is a momentary SPST (Single Pole Single Throw) switch that generates a board hardware reset. A hardware reset simultaneously resets the 602 processor, the Fujitsu MB86964 Ethernet controller and the National PC16552 serial communications controller. The SMI switch (U12) on the board is a momentary SPST switch that generates a system management interrupt interrupt on the 602 processor chip. The ROM Monitor supports using the system management interrupt as a mechanism for suspending the execution of an application. When debug is not used, the ROM Monitor simply passes the system management interrupt on to the application's first level interrupt handler.

**Table 6-4. 602 EVB Switches**

Location	Description
U2	Reset.
U12	Manual system management interrupt .

## 6.6 Power Supply

The board requires +5 volts at approximately 1 amp and +12 volts at 1 amp. The 12V supply supports the AUI Ethernet interface only. The power supply that is supplied with the 602 EVB is capable of supplying up to 4 amps. The power input connector is fused with Nanofuse devices made by Littlefuse corporation. The 602 EVB is crowbar protected on 5V supply. The 5V and 12V supplies are reverse bias protected. The pinout for the power connector can be found at "Power Connector" on page 5-18.

**Note:** It is recommended that the supply be connected to 120V prior to connection to the board.

## 6.7 Initialization

The 602 EVB contains a power on reset controller that provides a valid reset signal during powerup. Both Xilinx modules contain internal power sense capability and are forced to perform personality configuration during powerup. The Xilinx power sense circuitry is voltage rate sensitive. Additional loading of the power supply via the expansion connector might affect the power on rate, therefore the Xilinx is forced to configure by the reset controller.

The 602 processor may be reset by two other modes. The push-button reset (U2) provides a Hreset to the 602 but does not cause the Xilinx modules to be reconfigured. An Sreset may be issued through the RISCWatch JTAG connector. A Sreset only resets the 602 processor and has no effect on any of the peripheral components. A Sreset passes through the Xilinx U56 chip and clears the TEA.

At power up the DRAM controller blocks memory access for approximately 200us plus the time required to perform 8 refreshes (8\*16us more). Attempting DRAM access while the DRAM controller is blocking memory access will cause an exception as if the address was invalid. Either wait 500us after power up or read address 0x0 until a TEA exception no longer occurs.

Information regarding the reset and initialization of the PowerPC 602 processor can be found in the *PowerPC 602 RISC Microprocessor User's Manual*. If you require more information, refer to the osopen/opnebios/entry.s file.

## 6.8 Ethernet Controller

The Fujitsu MB86964 Ethernet controller's registers are addressed mapped from 0x0EA0 0007 through address 0x0EA0 FFFF. Ethernet reads and writes are performed though the BUF\_MEM\_PORT register, which is configured as 8 bytes. Example code that exercises the Ethernet controller can be found in the osopen/openbios/enetLib directory. Refer to the MB86964 data sheet for a detailed explanation of the controller's registers.

## 6.9 602 EVB Memory Map

The memory map of the 602 EVB is as follows:

PowerPC 602 EVB Memory Map	
0000 0000 -> 07FF FFFF	DRAM
0E80 0007 -> 0E80 FFFF	MK48T02 TOD Clock
0E90 0007 -> 0E90 FFFF	NS16C552 RS-232 Serial ports
0EA0 0007 -> 0EA0 FFFF	MB86964 Ethernet Controller
0EB0 0007 -> 0EB0 FFFF	Optrex DMC16117 LCD display
0EC1 0000 -> 0EC1 FFFF	IRQ Interrupt Controller
0FF0 0000 -> 0FF7 FFFF	Boot Flash, 512K
0FF8 0000 -> 0FFF FFFF	User Decode Space

## 6.10 IRQ Controller

The IRQ controller resides at address 0x0EC1 0003 in the memory map. Five data bits can be written to control the IRQ mask and four status bits can be read. The status bits reflect the condition of the interrupts from various devices. The fifth bit reports the condition of the overall mask (d27).

**Note:** Unlike other peripherals, the IRQ controller is wired to data lane 3.

The following are the interrupts that are fed to the IRQ controller and their corresponding mask bits:

**Table 6-5. IRQ Interrupts and mask bits**

Interrupt	Mask bit
Serial IRQ1	31
Serial IRQ2	30
Ethernet	29
External	28 (J1 pin b91)
OVERALL MASK	27

Writing a one (1) to any mask bit enables that function. Writing a zero (0) to 0/27 disables all interrupts from reaching the PowerPC 602 regardless of the status of the individual mask bits.

## 6.11 LCD Display

The Optrex DMC16117 LCD display contains two 8 bit addressable registers. The command register is located at 0x0EB00007 and the data register is located at 0x0EB0000F.



# 7

## 602 EVB ROM Monitor

---

---

This chapter describes the 602 EVB ROM Monitor program. This ROM resident program provides chip (and board level) initialization and a user interface menu that supports board diagnostics, program downloads, and debug.

### 7.1 ROM Monitor Source Code

The ROM Monitor source code is provided for ROM development purposes. This code is separate from the sample applications described in Chapter 8. The code is loosely organized by function in the following subdirectories and files within the **/usr/osopen/m602\_evb/openbios** directory (**\osopen\m602\_evb\openbios** for PC users):

- Makefile Top level makefile to create ROM monitor image (RS/6000 & SUN)
- makefile.mak Top level makefile to create ROM monitor image (PC)
- devTab.c Handles boot device definitions
- include/ C include files
- m4/ assembler preprocessor include files
- ppcLib/ C callable functions to access PowerPC special instructions
- enetLib/ Ethernet chip specific code
- ioLib/ I/O helper functions
- miscLib/ Miscellaneous routines used for ROM monitor
- s1Lib/ Serial Port interface routines
- s1ldLib/ Code to support S1 serial port downloads
- dbLib/ Ptrace debug interface routines
- emul\_h.s String and load/store float double handling for 602
- entry.s Processor and C environment initialization
- lib/ Repository for intermediate libraries
- netLib/ IP and UDP processing functions
- slipLib/ SLIP implementation
- align\_h.s Alignment handling code
- mapfile1 Mapfile to specify ROM Monitor linkage directives
- bios\_\*\*\*.map Load map of the ROM Monitor version \*\*\* shipped with the EVB
- flash/ Code to support re-programming the flash memory
- lcdLib LCD access functions

## 7.2 Communications Features

The 602 EVB ROM Monitor runs as part of the boot code in the flash memory on the board. The monitor communicates with an asynchronous terminal (or terminal emulator) attached to serial port 1 (SP1) on the EVB, through which the user accesses the monitor menu. The 602 EVB can download applications and communicate with the host debugger through serial port 2 (SP2) or the Ethernet adapter, depending on which devices are enabled. Communications between SP2 and the host use the Serial Link Internet Protocol (SLIP), while Ethernet communications use the Internet Protocol (IP) over standard Ethernet. The 602 EVB also supports the downloading of programs via serial port 1 (SP1). To use this feature, a VT100 terminal emulator that supports binary file transfers (such as kermit) must be used on the host system.

## 7.3 Bootp and tftp Configuration to support ROM Monitor Loads

Both the debugger and the ROM Monitor can be used to load applications onto the board. Details on how to use the debugger can be found in the *RISCWatch User's Guide*. To use the facilities of the ROM Monitor to download applications to the evaluation board, the host workstation must be configured to support the **bootp** protocol and **tftp** daemons. The configuration consists of two parts. The **bootptab** file on the host must be customized to match system requirements, and the **bootp** and **tftp** daemons (or servers) must be made available.

### 7.3.1 RS/6000 bootp and tftp configuration

To modify the **/etc/bootptab** file, you need to log in as **root** or the superuser (**su**). Entries describing the evaluation board to the host workstation must be added to this file. Complete details describing the bootptab file format are available in the *AIX Command Reference* under "bootpd". File entries suitable for our purposes are shown below:

```
slipc:hd=/usr/osopen/m602_evb/samples:bf=boot.img:bs:ip=8.1.1.5:sm=255.255.255.255  
enetc:ht=ethernet:hd=/usr/osopen/m602_evb/samples:bf=boot.img:bs:ip=7.1.1.5:sm=255.2  
55.255.255:ha=xxxxxxxxxxxx
```

Each of the entries, **slipc** and **enetc**, should be entered on a single line. The value of the **ethernet hardware address** field in the **enetc** entry, **ha=xxxxxxxxxxxx**, should match the twelve character hardware address listed for the Ethernet Boot Source on the ROM Monitor menu.

Both connections use the file **/usr/osopen/m602\_evb/samples/boot.img** as the source for the application image to be downloaded onto the board. Be sure that the **ht=ethernet** keyword is used for the Ethernet connection entry and that the IP addresses are those of the evaluation board. Note that the IP address in the **slipc** entry must match that of the IP address assigned to the board during serial port set-up. Since a board IP address was not required for Ethernet set-up, the IP address used in the **enetc** entry defines the IP address of the board for the Ethernet connection. If the suggested bootptab entries are used, 7.1.1.5 would be the board's Ethernet IP address. Take note of the board's IP addresses, since they must be made known to the ROM Monitor.

To start the **bootp** and **tftp** daemons on systems running AIX 3, do the following:

- log in as **root** or the superuser (**su**)
- enter **smit**
- select **Diskless Workstation Management and Installation**
- select **Start Daemons on Server**
- select **Start BOOTP Daemon**
- select **Do** or hit **Enter**

Upon successful completion, **bootp** configuration is complete. Continue for **tftp**:

- select **Done** or hit **PF3**
- select **Cancel** or hit **PF3** to return to the **Start Daemons on Server** screen
- select **Start TFTP Daemon**
- select **List**

If “**tftp udp**” is not on the list, **tftp** has already been started for the workstation. The configuration steps are complete. Select **Exit** to leave **smit**.

- select “**tftp udp**”
- select **Do** or hit **Enter**
- You should be at the **Add an inetd Subserver** screen. The defaults listed are acceptable.
- select **Do** or hit **Enter**

Upon successful completion, **tftp** configuration is complete. Select **Exit** to leave **smit**

To start the **bootp** and **tftp** daemons on systems running AIX 4, do the following:

- log in as **root** or the superuser (**su**)
- enter **smit**
- select **Processes and Subsystems**
- select **Subservers**
- select **Start a Subserver**
- select **bootps**
- select **OK**

Upon successful completion, bootp configuration is complete. Select **Done** and continue for **tftp**.

- select **Start a Subserver**
- select **tftp**
- select **OK**
- select **Done**

Upon successful completion, **tftp** configuration is complete. Select **Exit** to leave **smit**

### 7.3.2 PC bootp and tftp configuration

Not all TCP/IP packages include the bootpd and tftpd servers required for ROM Monitor downloads. For this reason both the bootpd and tftpd servers have been included in the EVB software package under the \osopen\bin directory. These servers can be installed and used in conjunction with Windows Socket compliant TCP/IP packages such as Trumpet Winsock and those that come with Windows 95 and Windows NT.

Since TCP/IP packages vary greatly, this section should be used only as a **guideline** for bootp and tftp set-up. Users should consult their TCP/IP documentation for specific details.

Configuration consists of two parts. The **bootptab** and **services** files on the host must be customized to match system requirements, and the bootpd and tftpd servers must be made available. If you choose to use the bootpd and tftpd servers provided with this package, you will need to modify your **autoexec.bat** file to specify the location of the bootptab and services files. This is accomplished by adding a line that sets up an ETC constant to the directory where the bootptab and services files are located (ie. SET ETC=C:\TRUMPET for Windows 3.1/Windows 95 Trumpet users, ETC=C:\WINDOWS for Windows 95 users, ETC=C:\WINNT35\system32\drivers\etc for Windows NT 3.51).

A sample bootptab file, **\osopen\PLATFORM\samples\bootptab.sam**, is included with the EVB software. This file can be copied to the ETC directory set in the autoexec.bat file and modified appropriately. Note that the bootptab file in the ETC directory must be named **bootptab** with no file extension. Entries describing the evaluation board to the host PC must be added to the bootptab file.

When creating or modifying the bootptab file, the following rules apply:

- blank lines and lines beginning with "#" are ignored
- each entry must be entered on a single line
- each entry must start with a hostname followed by the legends (see the sample bootptab file for legend descriptions)
- use ":" to separate each legend and leave no spaces between legends
- user must supply the host ip address via the "ip" legend
- if the "hd" (home directory) & "bf" (bootfile) legends are not provided for a particular entry, the first defined "hd" and "bf" legends in the bootptab file will be taken as default

File entries similar to those below would be suitable:

```
slipc:hd=\osopen\PLATFORM\samples:bf=boot.img:bs:ip=8.1.1.5:sm=255.255.255.255  
enetc:ht=ethernet:hd=\osopen\PLATFORM\samples:bf=boot.img:bs:ip=7.1.1.5:sm=255.255.  
.255.255:ha=xxxxxxxxxxxx
```

Each of the entries, slipc and enetc, should be entered on a **single** line. The value of the ethernet hardware address field in the enetc entry, ha=xxxxxxxxxxxx, should match the twelve character hardware address listed for the Ethernet Boot Source on the ROM Monitor menu.

Both connections use the file \osopen\PLATFORM\samples\boot.img as the source for the application image to be downloaded onto the board. Be sure that the **ht=ethernet** keyword is used for the Ethernet connection entry and that the IP addresses are those of the evaluation board. Note that the IP address in the slpc entry must match that of the IP address assigned to the board during serial port set-up. Since a board IP address was not required for Ethernet set-up, the IP address used in the enetc entry defines the IP address of the board for the Ethernet connection. If the suggested bootptab entries are used, 7.1.1.5 would be the board's Ethernet IP address. Take note of the board's IP addresses, since they must be made known to the ROM Monitor.

The **services** file (no file extention) must also exist in the ETC directory set in the autoexec.bat file. It must be updated with the port and protocol information for the bootpd and tftpd servers. To use the servers provided with this package, the following entries must be included in the services file:

bootps	67/UDP
bootpc	68/UDP
tftp	69/UDP

For the update to take effect, TCP/IP needs to be re-started. This may require a re-boot of the system and/or a restart of the TCP/IP package. After that, the bootpd and tftpd servers are ready for use.

### 7.3.2.1 Automatic startup for Windows 3.1 and Windows NT 3.51

Users may find it convenient to have the bootpd and tftpd servers brought up automatically when entering Windows. To do this for Windows 3.1, the bootpd and tftpd servers should be added to your Windows environment Startup window using the following procedure:

With Windows running, select the Program Manager and open the Startup window. Using the File pulldown menu on the Program Manager, select New to bring up a New Program Object window. From the New Program Object window, select Program Item and OK to open the Program Item Properties window. The Program Item Properties window requires that you provide Description, Command Line and Working Directory values. The following example shows one possible configuration.

Description: BOOTPD  
Command Line: BOOTPD -C D -H 7.1.1.4  
Working Directory: D:\OSOPEN\BIN

In the above example, the command line specifies how to invoke the bootpd server, and the working directory specifies where to find the bootpd server program (bootpd.exe). The -C parameter is used to specify a drive letter that is used in conjunction with bootptab file entries. Because the colon is used as a delimiter in bootptab file entries, the -C parameter is used as a mechanism by the bootpd server to concatonate a drive letter to the beginning of the hd: field. If the -C option is not specified, the current drive will be used as a default. The -H parameter is used to specify the ethernet or slip IP address of the host PC (set during host configuration) to the bootpd server.

Use the same procedure to set up the tftpd server. In this case, the Program Item Properties window entries will describe information used for the tftpd server. The following example shows a possible configuration:

Description: TFTPD

Command Line: TFTPD

Working Directory: D:\OSOPEN\BIN

If you do not wish to have the bootpd and tftpd servers run automatically upon entering Windows, they can be run individually from the Windows Program Manager, File, Run menu. Note that TCP/IP must be up and running before the servers can be run.

### **7.3.2.1 Automatic startup for Windows 95**

You may choose to run "BOOTPD.EXE" and "TFTPD.EXE" automatically every time that Windows 95 is started or you can run these programs only when needed. To make these program run automatically every time Windows 95 is started perform the following steps:

- Select 'Start' from the Windows 95 task bar.
- Select 'Settings'
- Select 'Taskbar'
- Select 'Start Menu Programs'
- Select 'Add...'
- In the command line field enter the following:  
    `BOOTPD -c C -h 7.1.1.4`  
(Where "C" is the driver letter containing the boot image and "7.1.1.4" is host IP address)
- Select 'Next'
- In the 'Select Program Folder' window, select the 'Programs/Startup' folder
- Select 'Next'
- Select 'Finished'
- To start "TFTP" follow the above steps, but enter the following in the command line field:

TFTPD

The BOOTP and TFTP demons will be started automatically upon the next restart of Windows 95.

### **7.3.3 SUN bootp and tftp configuration**

The Solaris and SunOS operating systems both provide a tftpd server but do not provide a bootpd server. For this reason a bootpd server has been included in the EVB software package under the /usr/osopen/bin directory.

A sample bootptab file, **/usr/osopen/PLATFORM/samples/bootptab.sam**, is included with the EVB software. This file should be copied to the **/etc** directory and renamed **bootptab** if a bootptab file does not already exist. You will need to log in as root or the superuser (su) to update or add files in the **/etc** directory. Entries describing the evaluation board to the host PC must be added to the bootptab file.

When creating or modifying the bootptab file, the following rules apply:

- blank lines and lines beginning with “#” are ignored
- each entry must be entered on a single line
- each entry must start with a hostname followed by the legends (see the sample bootptab file for legned descriptions)
- use “:” to separate each legend and leave no spaces between legends
- user must supply the host ip address via the “ip” legend
- if the “hd” (home directory) & “bf” (bootfile) legends are not provided for a particular entry, the first defined “hd” and “bf” legends in the bootptab file will be taken as default

File entries similar to those below would be suitable:

```
slipc:hd=/usr//osopen/PLATFORM/samples:bf=boot.img:bs:ip=8.1.1.5:sm=255.255.255.255  
enetc:ht=ethernet:hd=/usr/osopen/PLATFORM/samples:bf=boot.img:bs:ip=7.1.1.5:sm=255.  
255.255.255:ha=xxxxxxxxxxxx
```

Each of the entries, **slipc** and **enetc**, should be entered on a **single** line. The value of the ethernet hardware address field in the **enetc** entry, **ha=xxxxxxxxxxxx**, should match the twelve character hardware address listed for the Ethernet Boot Source on the ROM Monitor menu.

Both connections use the file **/usr/osopen/PLATFORM/samples/boot.img** as the source for the application image to be downloaded onto the board. Be sure that the **ht=ethernet** keyword is used for the Ethernet connection entry and that the IP addresses are those of the evaluation board. Note that the IP address in the **slipc** entry must match that of the IP address assigned to the board during serial port set-up. Since a board IP address was not required for Ethernet set-up, the IP address used in the **enetc** entry defines the IP address of the board for the Ethernet connection. If the suggested bootptab entries are used, 7.1.1.5 would be the board’s Ethernet IP address. Take note of the board’s IP addresses, since they must be made known to the ROM Monitor.

To start the **bootpd** and **tftpd** servers:

- log in as **root** or the superuser (**su**)
- ensure that the following entries are included in the **/etc/services** file:

bootps	67/udp
bootpc	68/udp
tftp	69/udp

- ensure that the tftp entry in the **/etc/inetd.conf** file is uncommented and modify as follows:

```
tftp    dgram  udp    wait    root    /usr/etc/in.tftpd    in.tftpd -s /
```

- add an entry for the bootpd server in **/etc/inetd.conf** as follows:

```
bootps dgram  udp    wait    root    /usr/osopen/bin/bootpd  bootpd -i
```

- reconfigure inetd for the updates made to the inetd.conf file. First find the process id for inetd :

```
ps -ef | grep inetd (Solaris)
```

```
ps -auex | grep inetd (SunOS)
```

Then send a hangup signal to reconfigure inetd:

```
kill -HUP <process id>
```

Bootp and tftp configuration is complete.

## 7.4 Accessing the ROM Monitor

The ROM Monitor expects a real or emulated VT100 type ASCII display attached to serial port 1 with line protocol parameters of 9600 baud, eight bits per character, no parity, and one stop bit. Once the terminal connected to SP1 is configured properly, you can access the ROM Monitor menu options, use the ping test, and load an application onto the evaluation board.

The ROM Monitor also provides the interface to the RISCWatch debugger. This facility, along with the image download process, is accessed via an IP network connection to the host workstation. Network configuration of the host was discussed earlier in the chapter on host configuration. The actual connection is either via SLIP (Serial Link Interface Protocol) running on serial port 2 at speeds up to 56K baud, or via standard Ethernet using the 10BaseT connector on the evaluation board.

## 7.5 ROM Monitor Operation

The ROM Monitor requires a block of DRAM for its operation and makes some assumptions about applications loaded on the board. Some of these assumptions may be disregarded if you do not need the ROM Monitor to interface with a debugger or otherwise support communication between the host workstation and the EVB.

Applications wishing to coexist with the ROM Monitor must observe the following constraints:

- Do not alter the Interrupt Base Register (IBR)
- Provide exception vectors for application events as if the IBR were set to 0x0002 0000. For example, an application's external interrupt handler should be located at 0x0002 0500. This is handled for you when using OS Open.

- Use storage addresses between 0x0002 3000 and the end of DRAM only, except for application vectors.
- Do not reset the IP bit (bit 25) in the MSR
- Do not start applications lower than address 0x0002 3000

The ROM Monitor is capable of handling most, but not all of the exceptions related to alignment. Alignment exceptions not handled by the ROM Monitor include the following:

- Alignment exceptions that occur while data translation is enabled are not handled by the ROM Monitor. When these exceptions occur, they are passed on to the application as alignment exceptions. In other words, if an application or operating system is running with data translation enabled, it must provide an interrupt handler to process alignment exceptions.
- On the 602, floating point load or store operations that involve double-precision operands that cannot be expressed as single-precision values, and operations producing denormalized numbers are handled by emulation software. The ROM Monitor alignment handler will attempt to emulate these operations, but in some cases may pass the exception on to the application as an alignment exception. An example would be when execution of an stfs (Store Floating-Point Single) instruction is attempted while the SP bit corresponding to the operand is improperly set (zero).

The ROM Monitor is capable of handling some, but not all of the exceptions related to emulation traps. Emulation trap exceptions not handled by the ROM Monitor include the following:

- On the 602, floating point load or store operations that involve double-precision operands that cannot be expressed as single-precision values, and operations producing denormalized numbers are handled by emulation software. Integer load and store string instructions are also handled by emulation software. The ROM Monitor will emulate the integer load and store string instructions and attempt to handle the floating point load and store operations. In some cases the ROM Monitor may pass the exception on to the application as an emulation trap exception. An example would be when execution of an stft (Store Floating-Point Single) instruction is attempted while the SP bit corresponding to the operand is improperly set (zero).

As a safe guard, the application or operating system may want to provide alignment and emulation trap exception handlers for floating point load and store operations. The ROM Monitor provides only partial coverage of these operations, and no coverage for operations where the SP and LT bits are improperly set.

Figure 7-1 shows the address map of the evaluation board under control of the ROM

Monitor.

ROM Monitor	0x0FF1 FFFF
	0x0FF0 0000
	⋮
IRQ Interrupt Controller	0x0EC1 0000
	⋮
LCD	0X0EB0 0000
	⋮
NIC	0x0EA0 0000
	⋮
DUART	0x0E90 0000
	⋮
RTC/NVRAM	0x0E80 0000
	⋮
	0x0080 0000 (end of DRAM)
Application Area	
Application Vectors	0x0002 3000
	0x0002 0000
ROM Monitor Area	
ROM Monitor Vectors	0x0000 2000
	0X0000 0000

Figure 7-1. ROM Monitor Address Map

## 7.6 Monitor Selections and Submenus

At this point it is assumed that the host has been properly configured, all board connections have been made, power has been supplied, and the terminal emulator running on the host has been configured and started successfully. The main menu, shown below, is displayed after the 602 EVB has been reset and the ROM Monitor completes initialization. Note that some of the values you see, in particular the ROM Monitor version, the IP addresses, and the ethernet controller's hardware address, may differ with those shown below.

Each menu option is described separately in the following sections. “Local” in the context of the ROM Monitor IP addressing means the IP address assigned to the evaluation board, while “remote” means the IP address assigned to the host workstation. Using option 8 to

save changes made to the configuration will allow the new values to persist beyond subsequent power-ons or resets. The ROM Monitor supports this by storing its configuration data in NVRAM.

### 7.6.1 Initial ROM Monitor Menu

The following menu is displayed after the board has been reset:

```
PPC602 2.1 ROM Monitor (8/2/96)
```

```
--- Device Configuration ---
```

```
Power-On Test Devices:
```

```
000 Enabled System Memory [RAM]
001 Enabled Ethernet [ENET]
004 Enabled Serial Port 2 [S2]
```

```
-----
```

```
Boot Sources:
```

```
001 Enabled Ethernet [ENET]
    local=7.1.1.5 remote=7.1.1.4 hwaddr=1000abcdef55
004 Enabled Serial Port 2 [S2]
    local=8.1.1.5 remote=8.1.1.4 hwaddr=ffffffffffff
005 Enabled Serial Port 1 [S1]
    Baud = 9600
```

```
-----
```

```
Debugger : Disabled
```

```
-----
```

- 1 - Enable/disable tests
- 2 - Enable/disable boot devices
- 3 - Change IP addresses
- 4 - Ping test
- 5 - Toggle ROM monitor debugger
- 6 - Toggle automatic menu
- 7 - Display configuration
- 8 - Save changes to configuration
- 9 - Set baud rate for s1 boot
- 0 - Exit menu and continue

```
>
```

## 7.6.2 Selecting Power-On Tests

Option 1 in the main menu selects power-on tests. These tests are run when the menu exits and before the ROM loader begins the bootp processing.

- 1 - Enable/disable tests
- 2 - Enable/disable boot devices
- 3 - Change IP addresses
- 4 - Ping test
- 5 - Toggle ROM monitor debugger
- 6 - Toggle automatic menu
- 7 - Display configuration
- 8 - Save changes to configuration
- 9 - Set baud rate for s1 boot
- 0 - Exit menu and continue

->1

When option 1 is selected, the following submenu is displayed:

--- ENABLE AND DISABLE POWER-ON TESTS ---

Power-On Test Devices:

- 000 Enabled System Memory [RAM]
- 001 Enabled Ethernet [ENET]
- 004 Enabled Serial Port 2 [S2]

-----  
select device to change ->

Selecting a test toggles its testing status. For example, since the System Memory test is enabled in the above menu, selecting 0 at the prompt disables it.

select device to change ->**0** [Selects system memory]

After the selection has been made, the new setting is displayed, followed by the main menu.

select device to change ->0

[RAM] test is disabled

[Message describing change]

--- Device Configuration ---

Power-On Test Devices:

- 000 Disabled System Memory [RAM]
- 001 Enabled Ethernet [ENET]
- 004 Enabled Serial Port 2 [S2]

-----  
Boot Sources:

001 Enabled Ethernet [ENET]

local=7.1.1.5 remote=7.1.1.4 hwaddr=1000abcdef55

004 Enabled Serial Port 2 [S2]

local=8.1.1.5 remote=8.1.1.4 hwaddr=ffffffffffff

005 Enabled Serial Port 1 [S1]

Baud = 9600

Debugger : Disabled

---

- 
- 1 - Enable/disable tests
- 2 - Enable/disable boot devices
- 3 - Change IP addresses
- 4 - Ping test
- 5 - Toggle ROM monitor debugger
- 6 - Toggle automatic menu
- 7 - Display configuration
- 8 - Save changes to configuration
- 9 - Set baud rate for s1 boot
- 0 - Exit menu and continue

->

Remember to use Option 8 to save any configuration changes that you may have made. If the changes are not saved, they will be lost upon an exit from the menu or upon a board reset.

### 7.6.3 Selecting Boot Devices

Option 2 in the main menu enables and disables boot devices.

- 1 - Enable/disable tests
- 2 - Enable/disable boot devices
- 3 - Change IP addresses
- 4 - Ping test
- 5 - Toggle ROM monitor debugger
- 6 - Toggle automatic menu
- 7 - Display configuration
- 8 - Save changes to configuration
- 9 - Set baud rate for s1 boot
- 0 - Exit menu and continue

->**2**

When option 2 is selected, the following submenu is displayed:

--- ENABLE AND DISABLE BOOT DEVICES ---

Boot Sources:

- 001 Enabled Ethernet [ENET]  
local=7.1.1.5 remote=7.1.1.4 hwaddr=1000abcdef55
- 004 Enabled Serial Port 2 [S2]  
local=8.1.1.5 remote=8.1.1.4 hwaddr=ffffffffffff
- 005 Enabled Serial Port 1 [S1]  
Baud = 9600

-----  
select device to change ->

Selecting a device toggles its boot status. Selecting 4, for example, would disable Serial Port 2 as a boot device.

select device to change ->**4**

[Selects serial port]

After the selection has been made, the new setting is displayed, followed by the main menu.

select device to change ->**4**

[S2] boot is disabled

[Message describing change]

--- Device Configuration ---

Power-On Test Devices:

- 000 Disabled System Memory [RAM]
- 001 Enabled Ethernet [ENET]
- 004 Enabled Serial Port 2 [S2]

-----  
Boot Sources:

- 001 Enabled Ethernet [ENET]  
local=7.1.1.5 remote=7.1.1.4 hwaddr=1000abcdef55
- 004 Disabled Serial Port 2 [S2]

```
local=8.1.1.5 remote=8.1.1.4 hwaddr=ffffffffffff
005 Enabled Serial Port 1 [S1]
Baud = 9600
-----
Debugger : Disabled
-----
1 - Enable/disable tests
2 - Enable/disable boot devices
3 - Change IP addresses
4 - Ping test
5 - Toggle ROM monitor debugger
6 - Toggle automatic menu
7 - Display configuration
8 - Save changes to configuration
9 - Set baud rate for s1 boot
0 - Exit menu and continue
->
```

When the user selects option 0 and exits from the monitor menu, the monitor attempts a boot of the application image on the host using the enabled boot sources in the order they are listed. In the above example, a boot would be attempted over Ethernet since it is the first boot source enabled. If more than one boot source is enabled, an attempt to boot over the first enabled device will be made. If that attempt fails, a boot over the next enabled device is attempted.

#### 7.6.4 Changing IP Addresses

Option 3 in the main menu allows users to change the IP addresses for the EVB and the host workstation. These addresses are used for bootp processing, debugger communications, and in the host connectivity “ping” test. **Note** - the local IP address is that of the board and the remote IP address is that of the host workstation. The IP addresses must match those set during host configuration.

- 1 - Enable/disable tests
- 2 - Enable/disable boot devices
- 3 - Change IP addresses
- 4 - Ping test
- 5 - Toggle ROM monitor debugger
- 6 - Toggle automatic menu
- 7 - Display configuration
- 8 - Save changes to configuration
- 9 - Set baud rate for s1 boot
- 0 - Exit menu and continue

->3

When option 3 is selected, the following submenu is displayed:

```
-- CHANGE IP ADDRESS --  
Device List:  
 001 Enabled Ethernet [ENET]  
           local=7.1.1.5 remote=7.1.1.4 hwaddr=1000abcdef55  
 004 Disabled Serial Port 2 [S2]  
           local=8.1.1.5 remote=8.1.1.4 hwaddr=ffffffffffff  
-----  
select device to change ->
```

Select the appropriate device:

select device to change ->1 [Selects Ethernet]

When a valid device is selected, the following submenu is displayed:

- 1 - Change local address
- 2 - Change remote address
- 0 - Return to main menu

->

Make the appropriate selection. To change the board’s IP address, you would select option 1, Change local address:

->1 [Selects the local address]  
Current IP address = (7.1.1.5) [Displays the current value]  
Enter new IP address ->Enter IP address in dot notation (e. g., 8.1.1.2)

Now enter the new IP address in dotted decimal notation:

#### 7.1.1.5

After the selection has been entered, the new configuration is displayed, followed by the main menu:

```
--- Device Configuration ---
Power-On Test Devices:
 000 Disabled System Memory [RAM]
 001 Enabled Ethernet [ENET]
 004 Enabled Serial Port 2 [S2]
-----
Boot Sources:
 001 Enabled Ethernet [ENET]
    local=7.1.1.5 remote=7.1.1.4 hwaddr=1000abcdef55
 004 Disabled Serial Port 2 [S2]
    local=8.1.1.5 remote=8.1.1.4 hwaddr=ffffffffffff
 005 Enabled Serial Port 1 [S1]
    Baud = 9600
-----
Debugger : Disabled
-----
1 - Enable/disable tests
2 - Enable/disable boot devices
3 - Change IP addresses
4 - Ping test
5 - Toggle ROM monitor debugger
6 - Toggle automatic menu
7 - Display configuration
8 - Save changes to configuration
9 - Set baud rate for s1 boot
0 - Exit menu and continue
->
```

This option should be repeated to set all of the IP addresses to their appropriate values. If the suggested IP addresses are being used, the local and remote addresses for both the Ethernet and the Serial Port should match those in the above menu. Remember to save any configuration changes via option 8.

## 7.6.5 Using the Ping Test

Option four in the main menu selects the ping test. The ping test can be used for a basic assurance test of IP connectivity to the host workstation. It should be performed after setting the IP addresses to insure host-to-EVB communications. If the ping test fails, users can not load applications on to the board. The local and remote addresses for the specified device are used for the source and destination of the ICMP ping packets.

- 1 - Enable/disable tests
- 2 - Enable/disable boot devices
- 3 - Change IP addresses
- 4 - Ping test
- 5 - Toggle ROM monitor debugger
- 6 - Toggle automatic menu
- 7 - Display configuration
- 8 - Save changes to configuration
- 9 - Set baud rate for s1 boot
- 0 - Exit menu and continue

->4

When option 4 is selected, the current configuration is displayed, followed by another command prompt:

```
--- PING TEST ---
Device List:
001 Enabled Ethernet [ENET]
    local=7.1.1.5 remote=7.1.1.4 hwaddr=1000abcdef55
004 Disabled Serial Port 2 [S2]
    local=8.1.1.5 remote=8.1.1.4 hwaddr=ffffffffffff
-----
select device to ping ->
```

Select the appropriate device to ping (in this case only Ethernet is enabled):

select device to ping ->1 [selects the Ethernet port]

If the board is able to successfully ping the host, a message similar to the following should appear:

```
Using [ENET] to ping. press any key to stop.
PING 7.1.1.4 56 data bytes
78 bytes from 7.1.1.4: icmp_seq=0 ttl=255 time=2 ms
78 bytes from 7.1.1.4: icmp_seq=2 ttl=255 time=1 ms
```

Hitting any key terminates the ping test. The main menu is redisplayed following the PING status report.

--- 7.1.1.4 ping statistics ---

2 packets transmitted, 2 packets received, 0% packet loss

- 1 - Enable/disable tests
- 2 - Enable/disable boot devices
- 3 - Change IP addresses
- 4 - Ping test
- 5 - Toggle ROM monitor debugger
- 6 - Toggle automatic menu
- 7 - Display configuration
- 8 - Save changes to configuration
- 9 - Set baud rate for s1 boot
- 0 - Exit menu and continue

->

If the ping test fails:

- Verify that the local and remote IP addresses are set correctly. The local IP address should be that of the board and the remote IP address should be that of the host. These IP addresses were assigned during host configuration (see earlier chapter).
- Verify that the cables are connected properly.
- Verify TCP/IP is running on the host.

**Note** - The ROM Monitor will not respond to an inbound ping test from the host unless the ROM Monitor is in Debug mode (via options 5 and 0) or the ROM Monitor ping test is active on the EVB at the same time (via option 4).

## 7.6.6 Entering the Debugger

Option 5 toggles the feature of the ROM Monitor that allows communication with the host based source level debugger. Debugging may be enabled/disabled, and saved as part of the configuration using option 8. The debugger is not actually called by the monitor until after the user exits the main menu by selecting option 0 (exit and continue):

```
--- Device Configuration ---
Power-On Test Devices:
 000 Disabled System Memory [RAM]
 001 Enabled Ethernet [ENET]
 004 Enabled Serial Port 2 [S2]
-----
Boot Sources:
 001 Enabled Ethernet [ENET]
    local=7.1.1.5 remote=7.1.1.4 hwaddr=1000abcdef55
 004 Disabled Serial Port 2 [S2]
    local=8.1.1.5 remote=8.1.1.4 hwaddr=ffffffffffff
 005 Enabled Serial Port 1 [S1]
    Baud = 9600
-----
Debugger : Disabled
-----
1 - Enable/disable tests
2 - Enable/disable boot devices
3 - Change IP addresses
4 - Ping test
5 - Toggle ROM monitor debugger
6 - Toggle automatic menu
7 - Display configuration
8 - Save changes to configuration
9 - Set baud rate for s1 boot
0 - Exit menu and continue
->5
ROM monitor debugger will be active on exit
1 - Enable/disable tests
2 - Enable/disable boot devices
3 - Change IP addresses
4 - Ping test
5 - Toggle ROM monitor debugger
6 - Toggle automatic menu
7 - Display configuration
8 - Save changes to configuration
9 - Set baud rate for s1 boot
0 - Exit menu and continue
->7
--- Device Configuration ---
Power-On Test Devices:
 000 Disabled System Memory [RAM]
```

```
001 Enabled Ethernet [ENET]
004 Enabled Serial Port 2 [S2]
-----
Boot Sources:
001 Enabled Ethernet [ENET]
    local=7.1.1.5 remote=7.1.1.4 hwaddr=1000abcdef55
004 Disabled Serial Port 2 [S2]
    local=8.1.1.5 remote=8.1.1.4 hwaddr=ffffffffffff
005 Enabled Serial Port 1 [S1]
    Baud = 9600
-----
Debugger : Enabled (on exit)
-----
1 - Enable/disable tests
2 - Enable/disable boot devices
3 - Change IP addresses
4 - Ping test
5 - Toggle ROM monitor debugger
6 - Toggle automatic menu
7 - Display configuration
8 - Save changes to configuration
9 - Set baud rate for s1 boot
0 - Exit menu and continue
->0
PowerPC ROM Monitor Debugger
```

```
Waiting for debug command...
Press any key to exit
```

Use option 8 to save the state of the ROM Monitor debugger. This option in combination with option 6, “Toggle automatic menu”, can be used to configure the EVB to automatically wait for the debugger to attach after power-on.

After enabling the ROM Monitor debugger (via option 5) and selecting option 0, the RISCWatch debugger can be started on the host and used to load an application onto the EVB. This is assuming the RISCWatch environment file has been updated for ROM Monitor communications. Once loaded successfully, the application can be run from the debugger.

The *RISCWatch Debugger User’s Guide* contains more information on how to use the debugger to load and execute files with the ROM Monitor as a non-JTAG target. At this point, it is recommended that users become familiar with the debugging environment by following the “Quick Start” sample debug session in the debugger’s User’s Guide. This session takes a user through the basics, including how to use the debugger to load and run applications on the board.

### **7.6.7 Disabling the Automatic Display**

Option 6 in the main menu disables the automatic monitor display when the EVB boots up. After option 6 has been selected and the configuration has been saved (via Option 8), the menu display is disabled but continues to function until the user exits from the main menu. Following the next power-on or reset, the menu is no longer automatically displayed. This allows the user's image to be downloaded automatically with no menu input required. This feature also allows a user to download an application with no cable connected to the serial port 1 on the EVB (that is, without a terminal emulator).

After the automatic menu display has been disabled, the main menu can be accessed (assuming a terminal emulator is attached successfully to SP1 on the EVB) by pressing any key during the first five seconds that the EVB is booting. Otherwise, application download processing starts without displaying the main menu.

## 7.6.8 Displaying the Current Configuration

Option 7 displays the current configuration.

- 1 - Enable/disable tests
- 2 - Enable/disable boot devices
- 3 - Change IP addresses
- 4 - Ping test
- 5 - Toggle ROM monitor debugger
- 6 - Toggle automatic menu
- 7 - Display configuration
- 8 - Save changes to configuration
- 9 - Set baud rate for s1 boot
- 0 - Exit menu and continue

->7

--- Device Configuration ---

Power-On Test Devices:

- 000 Disabled System Memory [RAM]
- 001 Enabled Ethernet [ENET]
- 004 Enabled Serial Port 2 [S2]

-----  
Boot Sources:

- 001 Enabled Ethernet [ENET]  
local=7.1.1.5 remote=7.1.1.4 hwaddr=1000abcdef55
- 004 Disabled Serial Port 2 [S2]  
local=8.1.1.5 remote=8.1.1.4 hwaddr=ffffffffffff
- 005 Enabled Serial Port 1 [S1]  
Baud = 9600

-----  
Debugger : Enabled (on exit)

- 1 - Enable/disable tests
- 2 - Enable/disable boot devices
- 3 - Change IP addresses
- 4 - Ping test
- 5 - Toggle ROM monitor debugger
- 6 - Toggle automatic menu
- 7 - Display configuration
- 8 - Save changes to configuration
- 9 - Set baud rate for s1 boot
- 0 - Exit menu and continue

->

When a menu operation is selected to alter configuration settings, the current configuration is automatically redisplayed.

### 7.6.9 Saving the Current Configuration

Option 8 saves the current configuration for subsequent power-ons/resets..

- 1 - Enable/disable tests
- 2 - Enable/disable boot devices
- 3 - Change IP addresses
- 4 - Ping test
- 5 - Toggle ROM monitor debugger
- 6 - Toggle automatic menu
- 7 - Display configuration
- 8 - Save changes to configuration
- 9 - Set baud rate for s1 boot
- 0 - Exit menu and continue

->8

Configuration has been saved

- 1 - Enable/disable tests
- 2 - Enable/disable boot devices
- 3 - Change IP addresses
- 4 - Ping test
- 5 - Toggle ROM monitor debugger
- 6 - Toggle automatic menu
- 7 - Display configuration
- 8 - Save changes to configuration
- 9 - Set baud rate for s1 boot
- 0 - Exit menu and continue

->

The configuration is saved in the NVRAM on the evaluation board and is retained until a new configuration is subsequently saved.

## 7.6.10 Setting the Baud Rate for S1 Boots

Option 9 provides a mechanism for setting the baud rate to be used by serial port 1 when it is used as a device to download programs. Downloading over serial port 1 requires the use of a VT100 terminal emulator that supports **kermit** binary file transfer over serial port 1. RS/6000 and Sun users should note that the TIP terminal emulator does not support kermit binary file transfers. Windows 3.1 users can use the Windows Terminal program to perform kermit binary file transfers, but the baud rate is limited to 19 200. Windows 95 users can use HyperTerminal to perform kermit file transfers at up to 115 200 baud. The kermit terminal emulator, available as shareware from the <http://www.columbia.edu/kermit> Internet site, can be used on any of the supported hosts to download programs over serial port 1 at speeds up to 115 200 baud.

--- Device Configuration ---

Power-On Test Devices:

- 000 Disabled System Memory [RAM]
- 001 Enabled Ethernet [ENET]
- 004 Enabled Serial Port 2 [S2]

---

Boot Sources:

- 001 Enabled Ethernet [ENET]  
local=7.1.1.5 remote=7.1.1.4 hwaddr=1000abcdef55
- 004 Disabled Serial Port 2 [S2]  
local=8.1.1.5 remote=8.1.1.4 hwaddr=ffffffffffff
- 005 Enabled Serial Port 1 [S1]

---

Debugger : Enabled (on exit)

---

- 1 - Enable/disable tests
- 2 - Enable/disable boot devices
- 3 - Change IP addresses
- 4 - Ping test
- 5 - Toggle ROM monitor debugger
- 6 - Toggle automatic menu
- 7 - Display configuration
- 8 - Save changes to configuration
- 9 - Set baud rate for s1 boot
- 0 - Exit menu and continue

->9

Select a baud rate for S1 boot

- 1 - 9600
- 2 - 19200
- 3 - 28800
- 4 - 38400
- 5 - 57600
- 6 - 115200

=>4

--- Device Configuration ---

Power-On Test Devices:

000	Disabled	System Memory [RAM]
001	Enabled	Ethernet [ENET]
004	Enabled	Serial Port 2 [S2]

---

Boot Sources:

001	Enabled	Ethernet [ENET]	local=7.1.1.5 remote=7.1.1.4 hwaddr=1000abcdef55
004	Disabled	Serial Port 2 [S2]	local=8.1.1.5 remote=8.1.1.4 hwaddr=ffffffffffff
005	Enabled	Serial Port 1 [S1]	Baud = 38400 [download baud rate appears here]

---

Debugger : Disabled (on exit)

---

- 1 - Enable/disable tests
- 2 - Enable/disable boot devices
- 3 - Change IP addresses
- 4 - Ping test
- 5 - Toggle ROM monitor debugger
- 6 - Toggle automatic menu
- 7 - Display configuration
- 8 - Save changes to configuration
- 9 - Set baud rate for s1 boot
- 0 - Exit menu and continue

->

Use Option 8 to save the selected speed after reset and power-on.

### 7.6.11 S1 Boot

To perform an S1 boot you must have a terminal emulator which supports kermit file transfer. The file must be a valid boot image and must be sent in binary mode. If you have selected to use a baud rate other than 9600, you must set the terminal emulator to run at that speed before loading the file and set the speed back to 9600 after the down-load is complete. The following example shows loading the “usr\_samp.img” file:

```
--- Device Configuration ---
Power-On Test Devices:
 000 Disabled System Memory [RAM]
 001 Disabled Ethernet [ENET]
 004 Disabled Serial Port 2 [S2]
-----
Boot Sources:
 001 Disabled Ethernet [ENET]
    local=7.1.1.5 remote=7.1.14 hwaddr=1000abcdef55
 004 Disabled Serial Port 2 [S2]
    local=8.1.1.5 remote=8.1.1.4 hwaddr=ffffffffffff
 005 Enabled Serial Port 1 [S1]
    Baud = 38400
-----
Debugger: Disabled
-----
1 - Enable/disable tests
2 - Enable/disable boot devices
3 - Change IP addresses
4 - Ping test
5 - Toggle ROM monitor debugger
6 - Toggle automatic menu
7 - Display configuration
8 - Save changes to configuration
9 - Set baud rate for s1 boot
0 - Exit menu and continue
->0
Booting from [S1] Serial Port 1...
```

PLEASE NOTE: You must now...

- a. Exit from terminal emulation mode
- b. Modify the baud rate of your host session
- c. Transmit a file to the target in binary mode
- d. Reset the host baud rate to 9600
- e. Reenter terminal emulation mode
- f. Hit enter to execute the downloaded program

At this point kermit users must get to the terminal emulator command mode and change the line speed to match what was selected by option 9 and tell the terminal emulator to send the file in binary format.

```
^l (Cntrl-\c)
(Back at waterdeep)
C-Kermit>set speed 38400
/dev/tty0, 38400 bps
C-Kermit>set file type bin
```

You can now load the file.

```
C-Kermit>send usr_samp.img
SF
Type escape character (^) followed by:
X to cancel file, CR to resend current packet
Z to cancel group, A for status report
E to send Error packet, Ctrl-C to quit immediately:
```

```
Sending: usr_samp.img => USR_SAMP.IMG
Size: 164864, Type: binary
```

```
.....
.....
.... [OK]
ZB
```

When loading is completed, you must change the baud rate back to 9600 bps before continuing.

```
C-Kermit>set speed 9600
/dev/tty0, 9600 bps
```

After setting the baud rate back to 9600 bps, re-connect to your terminal emulator and press enter to complete the down-load.

```
C-Kermit>con
Connecting to /dev/tty0, speed 9600.
The escape character is Ctrl-\ (ASCII 28, FS)
Type the escape character followed by C to get back,
or followed by ? to see other options
```

```
Loaded successfully ...
Entry point at 0x22f20 ...
```

```
Hello 602 user!
```

```
Your ROM Monitor version is : 2.1
```

```
Your 604 Evaluation Board has 33554432 bytes of DRAM installed.
```

```
Your Ethernet controller's network address is : 1000abcdef55
```

```
usr_samp done!
```

Assuming the S1 boot baud rate has been set to 38400 and option 0 has been selected to exit the ROM Monitor menu and initiate a load, Windows 95 HyperTerminal users can initiate the kermit binary file transfer by performing the following steps :

- Select **Call** and then **Disconnect**
- Select **File, Properties, Configure** and set the baud to match the baud rate set via ROM Monitor option 9. In this case, it is 38400.
- Select **OK** and **OK** again
- Select **Call** and then **Connect**
- Select **Transfer, Send File** and type the filename of the file to load. Set the **Protocol** to Kermit
- Select **Send**

Upon successful completion of the transfer, the baud rate must be changed back to 9600:

- Select **Call** and then **Disconnect**
- Select **File, Properties, Configure** and set the baud to 9600
- Select **OK** and **OK** again
- Select **Call** and then **Connect**
- Hit **Enter** to complete the down-load sequence

### 7.6.12 Exiting the Main Menu

Option 0 exits from the main menu, leaving the monitor active. If the debugger is active prior to selecting option 0, the ROM Monitor waits for the user to start the debugger on the host. In all other cases, option 0 initiates an attempt by the ROM Monitor to load an application from the host to the EVB over the enabled boot device(s). When downloading over the ethernet or SLIP (S2), the host bootp and tftp configuration must be completed for the ROM Monitor to load successfully. Once loaded successfully, the application is executed.

When serial port 1 is used, the ROM Monitor requires the user to follow additional instructions to complete the download. The example shown here describes the sequence required when programs are downloaded over serial port 1.

```
--- Device Configuration ---
Power-On Test Devices:
 000 Disabled System Memory [RAM]
 001 Disabled Ethernet [ENET]
 004 Disabled Serial Port 2 [S2]
-----
Boot Sources:
 001 Disabled Ethernet [ENET]
    local=7.1.1.5 remote=7.1.1.4 hwaddr=1000abcdef55
 004 Disabled Serial Port 2 [S2]
    local=8.1.1.5 remote=8.1.1.4 hwaddr=ffffffffffff
 005 Enabled Serial Port 1 [S1]
    Baud = 38400
-----
Debugger : Enabled (on exit)
```

---

```
1 - Enable/disable tests
2 - Enable/disable boot devices
3 - Change IP addresses
4 - Ping test
5 - Toggle ROM monitor debugger
6 - Toggle automatic menu
7 - Display configuration
8 - Save changes to configuration
9 - Set baud rate for s1 boot
0 - Exit menu and continue
->0
```

Booting from [S1] Serial Port 1...

PLEASE NOTE: You must now...

- a. Exit from terminal emulation mode
- b. Modify the baud rate of your host session
- c. Transmit a file to the target in binary mode
- d. Reset the host baud rate to 9600
- e. Re-enter terminal emulation mode
- f. Hit enter to execute the downloaded program

The ROM Monitor will now wait for you to follow the above steps. The idea is that you must temporarily modify the terminal emulation session baud rate to match the baud rate expected by the ROM Monitor for the serial port 1 download. The file must then be transferred to the EVB from the host. The baud rate is restored to 9600 so that terminal emulation support can function after the program has been downloaded. The ROM Monitor will wait for you to restore the baud rate (9600) and hit enter prior to executing the downloaded program. This prevents any program I/O from being lost or incorrectly displayed when it begins execution.

The following is an example of what you might see when the program is allowed to run:

```
Loaded successfully ...
Entry point at 0x23130 ...
```

```
.  
. .
```

## 7.7 ROM Monitor User Functions

The ROM Monitor contains several functions that are available to user programs. The prototypes of these functions can be found in the `usr_func.h` file in the `/usr/osopen/PLATFORM/include` directory (`\osopen\PLATFORM\include` for PC users). These functions include:

- `send_packet_on_bootdev()` - allows an IP packet to be sent over the device that was used to load the application program (either the ethernet or the second serial port, SP2).
- `sh_register()` - used to register a function that will be called when an IP packet is received by the ROM Monitor over the boot device.
- `get_board_cfg()` - reads the configuration data associated with the board.
- `enet_send_macframe()` - allows a frame to be sent over the ethernet.
- `enet_register()` - allows the user to register an IP address for the ethernet (an IP address different from that assigned to the ROM Monitor) and to specify a function to be called when a frame arrives for that address.
- `enetIsThere()` - determines if the ethernet chip is present on the board.
- `enetInit()` - initializes the ethernet.
- `getchar()` - reads one character at a time from the keyboard buffer over the first serial port (SP1).
- `s1putchar()` - writes one character to the first serial port (SP1).

Applications must follow a predefined protocol to access ROM Monitor user functions. An example showing the proper calling procedures are included in the `usr_samp.c` sample program in the `samples` directory. This sample program calls the `get_board_cfg()` ROM Monitor function to determine the amount of DRAM installed on the board. This program will be run as a sample program in the next chapter.

## 7.8 Flash Update Utility

The `openbios/flash` directory contains all the code you need to re-program the flash memory on the EVB. This utility takes a binary image file targeted for the ROM as input, and generates a loadable file that will re-program the flash memory with the data in the binary input file. The file can then be loaded by an existing ROM Monitor version (which will be over-written upon successful completion of the loaded program) or via RISCWatch JTAG.

**IMPORTANT: Please see the `readme.txt` file in the `openbios/flash` directory for important information regarding the use of this tool.**

Be aware that if you use the ROM Monitor bootp or the RISCWatch ROM Monitor mode download process to re-program the flash, and the program loaded contains errors that will not allow you to download images in the same manner, your flash may be corrupted and rendered useless. In this case you will need to use RISCWatch JTAG or a ROM burner to re-program the flash.

RISCWatch JTAG users will find a RISCWatch command file, `rw_flash.cmd` in the `openbios/flash` directory. This command file can be used to prepare the EVB, load the flash

update program containing the new binary image to program into the ROM, and start it running. This method can be used to program new flash parts, or to re-program a corrupted flash part when normal ROM Monitor downloads are not possible or inconvenient. When using this command file, RISCWatch **must** be used in JTAG mode.

# 8

## 602 EVB Sample Applications

---

This chapter describes the steps necessary to build and run the sample programs included in the 602 EVB software support package. This code is separate from ROM monitor code described in Chapter 7.

### 8.1 Overview

In the High C version of the EVB kit, the sample application programs are compiled, assembled, and linked using the IBM High C/C++ compiler, assembler, and linker. OS Open libraries are used during the link step to create an executable file in ELF format. This file includes the OS Open bootstrap code as well as other OS Open functions and is referred to as a boot file. One of the tools provided in the software support package, **eimgbld**, is then used to convert the boot file into the format used by the ROM Monitor to load programs onto the evaluation board (see Appendix B for more information on the ROM Monitor load format). The output of the **eimgbld** step is a file referred to as a boot image file.

There are several ways to load and execute a boot image file. One way is to use the ROM Monitor to load and execute the file. Network loads over Ethernet or SLIP require that the host contain the bootp and tftp servers and be properly configured to support the bootp and tftp protocols (see the previous chapters on host configuration and ROM Monitor setup).

Loads over serial port 1 require a terminal emulator that supports the kermit transfer protocol. A ROM Monitor load is initiated via option 0 from the ROM Monitor main menu.

Another way to load and execute the boot image file is to use the RISCWatch debugger in ROM monitor mode. To bring up RISCWatch in ROM Monitor mode (see the RISCWatch User's Guide for details), you must update the RISCWatch environment file for ROM Monitor communications, enable the ROM Monitor debugger (via option 5), exit the ROM Monitor menu (via option 0) and then start up RISCWatch on the host system. The RISCWatch **load image** command can then be used to load the boot image file onto the board. Once loaded successfully, the **attach 42** and **logoff** commands can be issued to execute the program. The **attach 42** command informs the ROM Monitor that a process will be running and the **logoff** command tells the ROM Monitor to exit debug mode and start the execution of the program. After program execution, users should quit and restart RISCWatch before loading another boot image file to run. Without quitting RISCWatch, subsequent boot image execution can not be guaranteed. (Note: RISCWatch also provides the means to load a boot file (as opposed to a boot image file) via its **load file** command. See the "Running Your Programs" section in the RISCWatch User's Guide for additional information. This section also describes the steps required to load and debug boot and boot image files.)

## 8.2 ROM Monitor Flash Image

The flash memory on the EVB comes preprogrammed with a specific version of the ROM Monitor. This version may not be latest version of the ROM Monitor. To run the samples in the software support package, the latest version should be used. The latest version of the ROM Monitor is included in the software support package in the file:

- **/usr/osopen/PLATFORM/openbios/lib/rom\_\*\*\*.img** (RS6K & SUN)
- **\osopen\PLATFORM\openbios\lib\rom\_\*\*\*.img** (PC)

where \*\*\* is equal to the ROM Monitor version. If the \*\*\* version number of the ROM Monitor in the software support package does not match the version number displayed by the monitor when it comes up on the board, you can load the more recent version of the monitor provided in the software support package to re-program the flash memory.

The **rom\_\*\*\*.img** file can be loaded using the ROM Monitor or the RISCWatch debugger. For it to load properly upon the selection of ROM Monitor option 0, it must be copied to **boot.img** if the suggested bootptab entry was used (see the previous chapter on bootp configuration).

To load using RISCWatch, enable the ROM Monitor debugger (via option 5), exit the ROM Monitor menu (via option 0), start RISCWatch on the host system (make sure the RISCWatch environment file is setup for ROM Monitor communications), then use the following RISCWatch commands to load and execute the **rom\_\*\*\*.img** image file:

- **load image /usr/osopen/PLATFORM/openbios/lib/rom\_\*\*\*.img** (RS6K & SUN)
- **load image \osopen\PLATFORM\openbios\lib\rom\_\*\*\*.img** (PC)
- **attach 42**
- **logoff**

You will see screen information similar to that shown below. Lines preceded by “\$\$” are annotation for this example and do not appear on the screen.

```
$$ Standard ROM Monitor load screen below
PPC602 1.2 ROM Monitor (9/5/95)
$$ Version 1.2 already installed corresponds to rom_12.img

--- Device Configuration ---
Power-On Test Devices:
 000 Disabled System Memory [RAM]
 001 Enabled Ethernet [ENET]
 004 Enabled Serial Port 2 [S2]
-----
Boot Sources:
 001 Enabled Ethernet [ENET]
    local=7.1.1.5 remote=7.1.1.4 hwaddr=1000abcdef55
 004 Disabled Serial Port 2 [S2]
    local=8.1.1.5 remote=8.1.1.4 hwaddr=ffffffffffff
```

```
004 Disabled Serial Port 1 [S1]
Baud = 38400
```

---

```
Debugger: Disabled
```

---

- 1 - Enable/disable tests
- 2 - Enable/disable boot devices
- 3 - Change IP addresses
- 4 - Ping test
- 5 - Toggle ROM monitor debugger
- 6 - Toggle automatic menu
- 7 - Display configuration
- 8 - Save changes to configuration
- 9 - Set baud rate for s1 boot
- 0 - Exit menu and continue

```
->0
```

```
$$ Selection of 0 causes evaluation board to be loaded. Previous
$$ arrangements must have been made to place the new ROM Monitor
$$ image (for ex. /usr/osopen/PLATFORM/openbios/lib/rom_13.img) in the
$$ place where bootp expects to find it (for ex. boot.img)
Booting from [ENET] Ethernet...
Sending bootp request ...
```

```
Loading file "/usr/osopen/PLATFORM/samples/boot.img" ...
Sending tftp boot request ...
Transfer Complete ...
Loaded successfully ...
Entry point at 0x23028 ...
```

```
$$ following information is from the ROM Monitor update program
#####
IBM 6XX Evaluation Kit FLASH Update #####
ROM Monitor Version 1.3
```

```
$$ The universally administered hardware address for the Ethernet
$$ controller is kept in the flash ROM and is displayed here.
$$ Do not change this value for normal ROM Monitor updates
Network Address =
1000abcdef55
```

```
Do you wish to change Network Address? (y or n) n
```

```
$$ Heed the following warning. The ROM Monitor image could be
$$ rendered unusable and the board useless until the flash ROM is
$$ replaced.
```

```
WARNING: You are about to re-program your ROM Monitor FLASH
```

image. Do NOT turn off power or press reset until this procedure is completed. Otherwise the card may be permanently damaged!!!

Do you wish to continue? (y or n)y

Verifying new FLASH Image...  
131072 matches, 0 mismatches

Update complete!  
All done!

### 8.3 Using the Software Samples

The sample application programs are in **/usr/osopen/PLATFORM/samples** (**\osopen\PLATFORM\samples** for PC users). It is recommended that users first build and run the Dhystone usr\_samp, and timestamp sample programs as detailed below, to become familiar with the working environment. These sample programs use **basic\_os.c** to provide a minimal OS Open configuration.

Additional details regarding the sample programs and application development in general can be found in the “Developing OS Open Applications” chapter in the OS Open User’s Guide. That chapter should be referenced for instructions on building and running the applprog, benchmk, mailsamp, and cat sample programs.

The sample makefile contains the directives needed to build all the sample programs. It is suggested that this makefile be used as the starting point for building subsequent user applications.

Before attempting to build the samples, ensure the **osopen/bin** directory and the directory that contains the compiler, are part of your execution path (these steps should be modified accordingly based on where the compiler and the software support package were actually installed):

For **RS/6000** and **SUN** hosts :

- issue the command:

```
export PATH=$PATH:/usr/osopen/bin:/usr/highcpc/bin
```

OR (to update your PATH permanently)

- Edit **~/.profile** using an editor such as **vi**.
- Add **PATH=\$PATH:/usr/osopen/bin:/usr/highcpc/bin** as a line in your profile before the line “**export PATH**”.
- Run **. ~/.profile** to update your profile.

For **PC** hosts:

- Edit **AUTOEXEC.BAT** using an editor such as **e** (you should back this file up before editing).

- If the following statement is missing, add it to the end of the file.  

```
SET PATH=C:\highcpc\bin;C:\osopen\bin;%PATH%;
```
- Run AUTOEXEC.BAT to update your path.

**NOTE:** The "make" utility supplied with your evaluation kit may not run under a Windows NT command prompt that is started by "cmd.exe". To avoid potential problems, start a DOS command prompt using the command "COMMAND.COM" and compile from there. Also, some Windows 95 users may receive a 'Program Requires MS-DOS Mode' pop-up message when compiling. To prevent this annoying message from occurring, select 'Properties' for the MS-DOS window you are compiling from, then select 'Advanced' and ensure that the 'Suggest MS-DOS mode as necessary' box is not checked.

### 8.3.1 Building and Running the Dhrystone Benchmark

The Dhrystone benchmark is a commonly available integer benchmark. Since the main loop of this benchmark fits into the caches of many processors, its validity as a predictor of system performance may be suspect. It is included here as an example of an application to be built, loaded onto the evaluation board, and executed.

To build the Dhrystone benchmark, enter the command "**make dhry**" from the command line while in the **samples** directory. The makefile will compile the Dhrystone source files, link the resulting object files with the support libraries, and produce the boot file, **dhry**, and the boot image file, **dhry.img**.

If the bootptab entry suggested in the chapter on "Host Configuration" was used, then **dhry.img** must be renamed or copied to **boot.img** in order to be selected by the ROM Monitor load process. Select option 0 from the ROM Monitor screen to load and run the image.

To load using RISCWatch, enable the ROM Monitor debugger (via option 5), exit the ROM Monitor menu (via option 0), start RISCWatch on the host system (make sure the RISCWatch environment file is setup for ROM Monitor communications), then use the RISCWatch **load image** command to load the **dhry.img** file. Once successfully loaded, issue the **attach 42** and **logoff** commands to return control to the ROM Monitor and initiate the run.

You should see the following messages (or ones like them) appear on the ROM monitor screen. Explanations preceded by ## do not appear on the screen but are added here as clarification.

```
Booting from [ENET] Ethernet...
Sending bootp request ...
## This requests the Host workstation to return the name of the boot image

Loading file "/usr/osopen/PLATFORM/samples/boot.img" ...
Sending tftp boot request ...
## Having obtained the file name, the ROM monitor uses tftp to retrieve the file from the
## host workstation
Transfer Complete ...
Loaded successfully ...
Entry point at 0x23a18 ...
## Having loaded an image, the ROM monitor is now transferring control to the application
## subsequent messages are from the application
```

```
Dhrystone Benchmark, Version 2.1 (Language: C)
Program compiled without 'register' attribute
Please give the number of runs through the benchmark:
```

At this point, enter the number of desired iterations. The test is designed not to give results if the selected iterations completes in less two seconds, so pick a large number ( $\geq 200000$ ). After the test completes, a check screen will be displayed, followed by the benchmark results. The results may vary based on the system environment.

### 8.3.2 Building and Running the **usr\_samp** Program

The **usr\_samp.c** program is included as a sample to be built and run on the EVB. It's a simple program that shows how to properly call the `get_board_cfg()` ROM Monitor user function to determine the ROM Monitor version, the amount of DRAM installed on the board and the Ethernet controller's MAC address. Developers interested in using any of the ROM Monitor user functions should use this program as a guide.

To build the **usr\_samp** program, enter the command “**make usr\_samp**” from the command line while in the **samples** directory. The makefile will compile the **usr\_samp.c** file, link the resulting object file with the support libraries, and produce the boot file, **usr\_samp**, and the boot image file, **usr\_samp.img**.

If the suggested bootptab was used, then **usr\_samp.img** must be renamed or copied to **boot.img** in order to be selected by the Rom Monitor load process. Select option 0 from the ROM Monitor screen to load and run the image.

To load using RISCWatch, enable the ROM Monitor debugger (via option 5), exit the ROM Monitor menu (via option 0), start RISCWatch on the host system (make sure the RISCWatch environment file is setup for ROM Monitor communications), then use the RISCWatch **load image** command to load the **usr\_samp.img** file. Once successfully loaded, issue the **attach 42** and **logoff** commands to return control to the ROM Monitor and initiate the run.

You should see the following messages (or ones like them) appear on the ROM Monitor screen.

```
Booting from [ENET] Ethernet...
Sending bootp request ...
```

```
Loading file "/usr/osopen/PLATFORM/samples/boot.img" ...
Sending tftp boot request ...
Transfer Complete ...
Loaded successfully ...
Entry point at 0x23e48 ...
```

```
Hello 602 user!
```

```
Your ROM Monitor version is : 2.1
```

```
Your 602 Evaluation Board has 8388608 bytes of DRAM installed.
```

```
Your Ethernet controller's network address is : 1000abcdef55
```

```
usr_samp done!
```

The DRAM amount listed should match the amount installed on the board.

### 8.3.3 Building and Running the timesamp Program

The **timesamp.c** program is included as a sample to be built and run on the EVB. This program is an example of how to properly time a particular function or benchmark. The user must know and define the time base frequency (the number of times the time base register is updated per second) in the timesamp.c to ensure the timing calculations are accurate.

To build the timesamp program, enter the command “**make timesamp**” from the command line while in the **samples** directory. The makefile will compile the timesamp.c file, link the resulting object file with the support libraries, and produce the boot file, **timesamp**, and the boot image file, **timesamp.img**.

If the suggested bootptab was used, then **timesamp.img** must be renamed or copied to **boot.img** in order to be selected by the Rom Monitor load process. Select option 0 from the ROM Monitor screen to load and run the image.

To load using RISCWatch, enable the ROM Monitor debugger (via option 5), exit the ROM Monitor menu (via option 0), start RISCWatch on the host system (make sure the RISCWatch environment file is setup for ROM Monitor communications), then use the RISCWatch **load image** command to load the **timesamp.img** file. Once successfully loaded, issue the **attach 42** and **logoff** commands to return control to the ROM Monitor and initiate the run.

You should see the following messages (or ones like them) appear on the ROM Monitor screen.

```
Booting from [ENET] Ethernet...
Sending bootp request ...

Loading file "/usr/osopen/PLATFORM/samples/boot.img" ...
Sending tftp boot request ...
Transfer Complete ...
Loaded successfully ...
Entry point at 0x23e48 ...
```

Please give the number of runs through the benchmark:

At this point, enter the desired number of runs through the function or benchmark being timed. In this sample, the function being timed should execute for approximately a second, so a number between 1 and 10 would suffice.

## 8.4 Resolving Execution Problems

Configuration errors in the network or bootp tables cause most of the problems with running the sample applications. This section contains information that will aid users in identifying common problems.

### 8.4.1 Using the Ping Test on the ROM Monitor to Verify Connectivity

If the ping test fails, verify that TCP/IP is running on the host system and that the IP addresses on the selected interface are correct. The local address refers to the IP address of the evaluation board, and the remote refers to the host workstation address. The host workstation address must match the one selected during configuration of the host network interface. Also consult your TCP/IP documentation to insure proper network configuration.

### 8.4.2 bootp and tftp Servers (Daemons) for ROM Monitor loads

Insure that the bootp and tftp servers are started on the host workstation. If possible, use the **tftp** command from another workstation to retrieve the load image. If this fails, make sure the image exists in the target directory and that it is readable by "others". If the tftp transfer succeeds, check the bootptab entry in the **bootptab** file to insure that it specifies the correct interface and IP address of the evaluation board.

## 8.5 Using OS Open Functions

OS Open provides the following major classes of functions for the embedded programming environment:

- Thread management

The unit of execution context for OS Open is the thread as defined by POSIX standards. Functions are provided to create threads with various scheduling and execution attributes. To manage the execution environment, serialization and synchronization primitives are part of OS Open. The system also provides functions to associate data with specific threads.

- Storage management

OS Open supports variable block allocations in the form of a heap. Functions are provided to extend the heap, query heap usage, and allocate storage to meet alignment constraints. OS Open also provides an independent storage management mechanism to allocate fixed blocks of storage in constant time.

- Interrupt and fault support

OS Open provides functions to attach user-written code to any of the processor exceptions and interrupts. Most of the functions of OS Open can be used in these interrupt handlers, except for those functions that suspend execution or are valid only in the context of an executing thread. When the underlying hardware platforms support it, OS Open platform-specific libraries provide additional functions to attach user-written code to external interrupts supported on the platforms.

- Clock and timer management

OS Open functions provide time-of-day clock support and the ability to create, use, and destroy timers. These timers can be one-time or periodic.

- Device support

OS Open functions support the installation of user-written device drivers to provide character special files, block special files, and logical file systems. Low-level POSIX I/O (read, write) as well as ANSI C stream (fget, fput) functions are provided for device and regular file access.

- ANSI C library support

OS Open provides a comprehensive set of ANSI C functions, providing support for string manipulation, memory management, string-to-number conversion, input/output, nonlocal jumps, and variable arguments.

- Pseudo device driver support

OS Open provides several functions, such as TTY and DOS file system functions, that are installed and managed like device drivers, but they do not manipulate actual hardware nor do they have platform or device dependencies.

OS Open provides functions that create and manage TCP/IP sockets. Network interface functions for Token Ring, Ethernet, and Serial Line Interface Protocol (SLIP) are also provided. With the TCP/IP protocol stack and network interfaces, additional functions are provided that implement several popular networking utilities, such as ping, ifconfig, ftp, and telnet.

- Debug functions and kernel abstract data types

OS Open provides functions that set, clear, and query breakpoints. OS Open features an internal circular trace buffer for operating system and user events. Also, functions are provided that dump kernel data objects in a readable form.

# Application Libraries and Tools

This chapter describes some of the application libraries and tools available in the EVB software support package. See the OS Open *User's Guide* and *Programmer's Reference* for additional information.

## 9.1 OS Open Libraries

The OS Open operating system comprises a real-time executive and optional libraries of functions and macros.

The real-time executive provides a operating system core for embedded applications. Depending on an application's requirements, an embedded application may also incorporate one or more optional libraries.

This modular approach enables embedded system developers to scale an OS Open operating system to match their application requirements. Because unneeded features are not present, an OS Open configuration can provide savings in system hardware, initialization and reset time, and program size.

Table 9-1 summarizes the OS Open libraries, described in the OS Open User's Guide and in this user's guide. For detailed descriptions of the OS Open functions and macros, refer to the *OS Open Programmer's Reference*.

**Table 9-1. OS Open Libraries**

Library	File Name	Platforms
Alignment Exception Support Library	alignLib.a	Common
ANSI C Library	cLib.a	Common
ANSI C Math Library	mathLib.a	Common
ANSI C I/O Library	fsLib.a	Common
Block Buffer Library	bbuffLib.a	Common
Bios Ethernet Library	benetLib.a	602 EVB
Boot Library(DRAM)	bootLib.a	602 EVB
ROM Monitor Ethernet Interface Library	benetLib.a	602 EVB

**Table 9-1. OS Open Libraries**

<b>Library</b>	<b>File Name</b>	<b>Platforms</b>
C++ runtime support (High C++™ support) Library	cppLib.a, crt.i.o, crtn.o, mwdctor.o	ELF
Card Services/enabler software layer for PCMCIA support	csLib.a	Common
Clock Support Library and NV-RAM	clockLib.a	602 EVB
Debug Support Library	dbLib.a	Common
Device and File Support Library	devLib.a	Common
DOS File System Support Library	fatLib.a	Common
Dynamic Loader Library	ldrLib.a	Common
Ethernet Support Library	enetLib.a	602 EVB
File Transfer Protocol Support Library	ftpLib.a	Common
Floating Point Library	fpeLib.a	Common
Input/output Support Library	ioLib.a	602 EVB
Kernel Abstract Data Types Library	kadtLib.a	Common
LCD Library	lcdLib.a	602 EVB
Network Support Library	netLib.a	Common
NFS Support Library	nfsLib.a	Common
OpenShell	shell.o	Common
PCMCIA ATA/IDE Hard disk device driver	pataLib.a	Common
PowerPC Low Level Access Support Library	ppcLib.a	602 EVB
Queue Library	queLib.a	Common
RAM Disk Library	ramdLib.a	Common
Rate Monotonic Scheduling (RMS) Library	rmsLib.a	Common
Remote Source Level Debug Library	rsldLib.a	Common
Ring Buffer Library	rngLib.a	Common
RPC Support Library	rpcLib.a	Common
Runtime Library	runlib.a	Common
SCSI Support Library	scsiLib.a	Common
Serial Support Library	asyncLib.a	602 EVB

**Table 9-1. OS Open Libraries**

Library	File Name	Platforms
Socket Services for PCMCIA support	ssLib.a	Common
Symbol Support Library	symLib.a	Common
TCP/IP Protocol Support Library	tcpipLib.a	Common
Telnet Daemon Support Library	tntedLib.a	Common
Telnet Client Support Library	telnet.o	Common
The Real-time Executive	rtx.o, rtxLib.a	Common
OS Open Minimal Kernel	rtxmin.o	Common
OS Open Kernel Extensions for the minimal kernel	rtxext.o	Common
Timer Tick Support	tickLib.a	602 EVB
Trivial File Transfer Protocol	tftp.o	Common
TTY Support Library	ttyLib.a	Common

The real-time executive, the only required component in an OS Open operating system, provides a full set of basic operating system services:

- Thread management
- Storage management
- Signals
- Clocks and timers
- Interrupt and fault handling
- Message queues
- Semaphores
- Trace buffer support
- Miscellaneous services

The C functions for the real-time executive functions are in two libraries, **rtx.o** and **rtxLib.a**. The **rtx.o** library contains the OS Open real-time executive. The **rtxLib.a** library contains interface routines to OS Open functions, and is linked with application programs to resolve calls to the real-time executive.

## 9.2 Using Libraries and Support Software

The object libraries specific to the 602 EVB are described below:

**Table 9-2. OS Open Libraries for the 602 EVB**

Library	File Name
Boot Library	bootdLib.a
Ethernet Device Driver Support Library	enetLib.a
Input/Output Support Library	ioLib.a
LCD Library	lcdLib.a
PowerPC Low Level Access Support Library	ppcLib.a
Real-time Clock Interface Support Library with NV-RAM	clockLib.a
ROM Monitor Ethernet Interface Library	benetLib.a
Serial Support Library	asyncLib.a
Software Timer Tick Support Library	tickLib.a

### 9.2.1 Serial Port Support Library

This library supports the serial ports on the 602 EVB. Use in conjunction with the function provided by **devLib.a** and **fsLib.a** to provide a high level I/O interface to application programs. The serial port support functions reside in the **asyncLib.a** library.

### 9.2.2 Boot Library(RAM)

This library contains the OS Open bootstrap program for the appropriate platform. The boot library performs initial processing to prepare the completed application program for execution on the platform. For the 602 EVB, this processing includes stripping the header from the completed image and moving the loaded program such that real addresses correspond with addresses assumed by the language development tools. The boot library for the 602 EVB also dynamically determines available heap space and prepares the symbol table for use by OS Open symbol management routines. The boot library does not export any functions.

### 9.2.3 Input/Output Support Library

The input/output functions reside in the **ioLib.a** library. To initialize the I/O subsystem, you must call **ioLib\_init()** (normal mode) or **dbg\_ioLib\_init()** (ROM Monitor debug/ethernet) before performing any I/O other function.

## **9.2.4 PowerPC Low-Level Processor Access Support Library**

The low-level access support library contains C-callable versions of the special PowerPC instructions. A few of the sample programs use these functions to manipulate the PowerPC 602's special registers. These functions provide access to processor instructions not generated by compilers. For example, device drivers often have a requirement to control data caching, disable interrupts, synchronize I/O, and other processor and platform-specific operations. The low-level access support functions reside in the **ppcLib.a** library.

## **9.2.5 ROM Monitor Ethernet IP Interface Library**

This library contains routines allowing access to the ROM Monitor's Ethernet IP interface. These functions allow the Ethernet to be simply configured with a unique IP address for use with TCP/IP functions. The ROM Monitor Ethernet IP Interface functions reside in **benetLib.a** library.

## **9.2.6 Real-time Clock Interface Support Library**

This library contains routines to read and set the 602 EVB battery-backed real-time clock. These functions are not to be confused with the real-time clock functions provided directly by OS Open when the system is running. The real-time clock interface support functions reside in the OS Open's **clockLib.a** library and are available to perform the following features:

- Set the OS Open clock from the real-time clock.
- Set the real-time clock from user-supplied data.
- Calibrate the real-time clock chip.
- Read and write NV-RAM in the clock chip.

## **9.2.7 Integrated Ethernet IP Interface**

This library provides the support for packet level interface to the integrated Ethernet interface in the 602 EVB. The Ethernet port support functions reside in the **enetLib.a** library.

## **9.2.8 Software Timer Tick Support Library**

The OS Open system requires a periodic call to **timertick\_notify()** to maintain internal clocks and timer functions. The **tickLib.a** library contains an implementation of the **timertick\_notify()** function for PowerPC architecture machines. Timer tick support functions reside in the **tickLib.a** library.

## 9.3 Device Drivers Supplied with the 602 EVB

Device drivers provided with the 602 EVB include:

- Asynchronous
- Ethernet
- ROM Monitor Ethernet
- Liquid Crystal Display (LCD)

Examples and references are provided where appropriate.

For more information about any of the OS Open functions mentioned in this chapter, refer to the *OS Open Programmer's Reference*.

### 9.3.1 Asynchronous Device Driver

The asynchronous device driver supports the two asynchronous communication ports found on the . Following is a brief functional description of the device driver:

- Support from 50 baud.
- Full duplex modem line control discipline.
- Overrun error, parity error, and framing error detection.
- BREAK interrupt detection.
- Support for data length of 5, 6, 7, and 8 bits.
- Support for 1, 1.5 and 2 stop bits.
- Support for receive and transmit parity.
- Support for odd and even parity.
- Support for transmitting BREAK.
- Support for 16 byte FIFO in the universal asynchronous receiver transmitter (UART).
- Programmed I/O (PIO) interrupt-driven slave communication.
- Interrupt driven input/output.
- Polled output functions.

Since only full duplex modem line control discipline is supported, connection between the asynchronous port and another device must be made through a "NULL" modem. A NULL modem is a device that crosses transmitted data and received data pins to enable communication. The only time a NULL modem is not necessary is when connection is made to a real modem device.

#### 9.3.1.1 Device Driver Installation

The asynchronous device driver is installed by calling **driver\_install()**. Following is an example of asynchronous device driver installation:

```
#include <sys/asyncLib.h>
int devhandle;
event_t event=10;
rc=driver_install(&devhandle, async_init, event);
```

The parameter *event* specifies the external interrupt id and must be between DOGWOOD\_EXT\_MIN and DOGWOOD\_EXT\_MAX.

**async\_init()** is declared in the file **<sys/asyncLib.h>** as follows:

```
int async_init(driver_t *dsw, va_list args)
```

Upon successful installation, **driver\_install()** returns 0; otherwise –1 is returned. For more information on **driver\_install()**, refer to the *OS Open Programmer's Reference*.

### 9.3.1.2 Device Installation

After the asynchronous device driver is installed, named devices can be created using **device\_install()**. Following is an example of device installation.

```
#define S1DB_PARMS 3686400, (unsigned char *) 0x0E90047, 8, 0x0  
rc=device_install("/dev/s0", CHRTYPE, devhandle, 1, 128, 128,S1DB_PARMS);
```

For device installation, *devhandle* is the value obtained from the **driver\_install()**. Device type CHRTYPE is defined in **<sys/devDrivr.h>**.

Additional parameters passed in the **device\_install()** call are as follows:

Parameter	Meaning
Fourth Parameter	Port number to be installed (1)
Fifth Parameter	Size of write buffer
Sixth Parameter	Size of read buffer
Seventh Parameter	Input clock for the divisor
Eight Parameter	UART base register address 0x8003F8 or 0x8002F8
Ninth Parameter	UART register address delta, always 1
Tenth Parameter	Interrupt DOGWOOD_EXT_MIN < event <DOGWOOD_EXT_MAX (0 < event <3)

These are positional parameters.

**Note:** Write and read buffer sizes indicate number of characters that can be buffered in the device driver.

Upon successful installation, **device\_install()** returns 0; otherwise –1 is returned. When the device is installed, error reporting for the device is turned off and xon/xoff pacing is enabled. For more information on **device\_install()**, refer to the *OS Open Programmer's Reference*.

### 9.3.1.3 Opening Asynchronous Communication Ports

After the device is installed, the **open()** system call can be used to open a particular device. Following is an example of the **open()** system call used against the asynchronous port:

```
fd1=open("/dev/s0", O_RDWR, asyncParityNone, asyncParityOdd,  
        asyncStopBits1, asyncDataBits8, 9600);
```

Additional parameters passed in **open()** are as follows:

Parameter	Meaning
First Parameter	Check/generate parity flag. Valid values are: asyncParityNone and asyncParityGen_Check
Second Parameter	Parity type. Valid values are asyncParityEven and asyncParityOdd. Because parameters are positional, this parameter must be specified even if parity is not used.
Third Parameter	Number of stop bits. Valid values are asyncStopBits1, asyncStopBits15 and asyncStopBits2. One and a half stop bits are only valid for data length of 5.
Fourth Parameter	Data length. Valid values are asyncDataBits5, asyncDataBits6, asyncDataBits7, and asyncDataBits8.
Fifth Parameter	Baud rate. Valid values range from 50 baud.

These are positional parameters. All parameter constants can be found in **<sys/iocctl.h>**.

**Note:** The *oflag* parameter, O\_RDWR in this example, which is passed in the open call, is ignored by the device driver. When successful, **open()** returns a file descriptor, otherwise -1 is returned. **open()** can be called multiple times against the same asynchronous port.

Communication parameters passed during the last **open()** call are set in the asynchronous port. For more information on **open()**, refer to the *OS Open Programmer's Reference*.

### 9.3.1.4 Reading and Writing

After successfully installing and opening the asynchronous port, **read()** and **write()** calls can be issued against that port. Multiple threads can issue **read()** and **write()** calls to the same port at the same time. However, simultaneous **read()** calls issued to the same port may block or be processed in an unexpected order. For these instances, thread scheduling and synchronization must be handled by the application.

Following is an example of **read()** and **write()** calls:

```
rc=write(fd1,"nOS Open Real-time Executive\n", 29);  
rc=read(fd1, buffer, 10);  
fd1 is the value obtained from the open() call.
```

**Note:** For more information on **read()** and **write()**, refer to the *OS Open Programmer's Reference*.

### 9.3.1.5 I/O Control

An **ioctl()** call issued against asynchronous device driver accepts the commands listed in Table 9-3. All parameter constants can be found in **<sys/ioctl.h>**

**Table 9-3. ioctl() Commands for Asynchronous Device Drivers**

Command	Parameters	Explanation
ASYNCBAUDSET	Value from 50	Sets baud rate
ASYNCAUDGET	Pointer to integer	Returns baud rate
ASYNCTRIGSET	asyncFifoTrigger1, asyncFifoTrigger4, asyncFifoTrigger8, asyncFifoTrigger14	Sets FIFO trigger level for asynchronous port
ASYNCTRIGGET	Pointer to integer	Returns current trigger level
ASYNCBREAKSET	None	Starts sending BREAK on port
ASYNCBREAKCLR	None	Stops sending BREAK on port
ASYNCDSTICKGET	Pointer to integer	Returns the way the parity bit is interpreted by the port
ASYNCDSTICKZERO	None	Disables stick parity
ASYNCDSTICKONE	None	Parity interpretation tracks even/odd parity
ASYNCRERRORGET	Pointer to integer	Returns and clears read error conditions. Values are defined in <code>asynclib.h</code>
ASYNCWERRORGET	Pointer to integer	Returns and clears write error conditions. Values are defined in <code>asynclib.h</code>
ASYNCCERROREN	None	Enables error reporting
ASYNCCERRORDIS	None	Disables error reporting. All pending errors are cleared
ASYNCCERRORGET	Pointer to integer	Returns error reporting enabled flag
ASYNCDLENGET	Pointer to integer	Returns current data length
ASYNCDLENSET	asyncDataBits5, asyncDataBits6, asyncDataBits7, asyncDataBits8	Sets data length
ASYNCSTOPGET	Pointer to integer	Returns number of stop bits

**Table 9-3. ioctl() Commands for Asynchronous Device Drivers**

Command	Parameters	Explanation
ASYNCSTOPSET1	None	Sets number of stop bits to 1
ASYNCSTOPSET1_5	None	Sets number of stop bits to 1.5
ASYNCSTOPSET2	None	Sets number of stop bits to 2
ASYNCPARITYNONE	None	Disable parity
ASYNCPARITYGEN	None	Enable parity
ASYNCPARITYSGET	Pointer to integer	Return parity status (enabled/disabled)
ASYNCPARITYODD	None	Sets parity to odd
ASYNCPARITYEVEN	None	Sets parity to even
ASYNCPARITYGET	Pointer to integer	Returns parity type
ASYNCXONENABLE	None	Enables XON/XOFF flow control
ASYNCXONDISABLE	None	Disables XON/XOFF flow control
ASYNCXONGET	Pointer to integer	Returns XON/XOFF flow control status
ASYNCMODEMSTAT	Pointer to integer	Returns modem status
ASYNCFLUSHIN	None	Flushes input buffer
ASYNCFLUSHOUT	None	Flushes output buffer
ASYNCDRAIN	None	Blocks until all characters in output buffer have been transmitted
ASYNCIGNBREAK	None	Ignores break interrupts
ASYNCSIGBREAK	None	Sends SIGINT on reception of break condition
ASYNCERRBREAK	None	Returns error from read upon reception of break condition. 0x00 is placed in the receive buffer at the position where break occurred.

Following is an example of an **ioctl()** call issued against an asynchronous device:

```
rc=ioctl(fd1, ASYNCXONDISABLE);
if (rc !=0) printf("ioctl failure\n");
fd1 is the value obtained from the open() call.
```

### 9.3.1.6 Polled Asynchronous I/O

A function is provided for polled output to s1 and s2 serial ports:

```
int s1dbprintf(unsigned long uart_clock, unsigned char *base_reg, int reg_delta,  
event_t event, const char *format, ...)  
int s2dbprintf(unsigned long uart_clock, unsigned char *base_reg, int reg_delta,  
event_t event, const char *format, ...)
```

The parameters passed to these functions are identical to **printf()** except for *uart\_clock*, *base\_reg*, *reg\_delta*, and *event*. *uart\_clock* specifies the clock speed, *base\_reg* specifies the address of the base UART register, *reg\_delta* specifies the address space between UART registers, and *event* specifies the external interrupt level. Because polled I/O transmits characters synchronously, these functions may be called from first level interrupt handlers (FLIHs) or a user-supplied panic function. Since the function waits until the characters are actually sent before returning, use of this with long strings can significantly affect the timing of calling programs.

### 9.3.2 Ethernet Device Driver

The Ethernet device driver is a character device driver supporting packet level read/writes to the integrated Ethernet controller. The driver features the ability to open multiple files. Each file receives packets for a specific standard Ethernet or 802.3 address.

Function highlights are:

- Up to 8 receive channels
- Size of receive buffer pool determined by user at driver install time.

#### 9.3.2.1 Device Driver Installation

The Ethernet device driver is installed by calling the **driver\_install()** function. Following is an example of Ethernet device driver installation.

```
#include <enet.h>
int devhandle;
char mac_array[6];
rc=driver_install(&devhandle, enet_init,
enet_intlvl, /* Interrupt level, typically 2*/
register_base,          /* NIC base register address, typically 0xEA000007*/
register_delta,/* NIC register spacing, typically 8 */
num_blocks,           /* number of 256 byte buffers allocated for driver use */
sram_size,/* must be 8 or 32, typically 8*/
mac_array);          /* location of universal MAC address */
```

*mac\_array* points to the 6 byte ethernet hardware address. Typically this value resides in ROM between the strings “\*NA” and “\*END” following address 0xFFFFE00.

Upon successful installation, **driver\_install()** returns 0; otherwise -1 is returned. For more information about the **driver\_install()** function, refer to the *OS Open Programmer’s Reference* and the OS Open samples thread0.c file.

#### 9.3.2.2 Device Installation

After the Ethernet device driver is installed, Ethernet devices can be installed using the **device\_install()** function. Following is an example of device installation:

```
rc=device_install("/dev/en0", CHRTYPE, devhandle);
```

For device installation, *devhandle* is the value obtained from the **driver\_install()**. Device type CHRTYPE is defined in **<sys/devDrivr.h>**.

Upon successful installation, **device\_install()** returns 0; otherwise -1 is returned. At this point, files may be opened against the Ethernet device.

#### 9.3.2.3 Opening and Closing Ethernet Files

After the device is installed, the **open()** system call can be used to open a particular device. Following is an example of the **open()** system call used to open an Ethernet port:

```
fd1=open("/dev/en0", O_RDWR);
```

When successful, **open()** returns the open file descriptor; otherwise -1 is returned. **open()** can be called multiple times against the same Ethernet device.

When using the **close()** function, the call to the driver-specific **close()** is deferred until all open files on the device are closed. This means that when an Ethernet file is closed, the channel address associated with the file will not be freed if another Ethernet file is open. Be aware that if the Ethernet interface has been connected to the TCP/IP protocol stacks via **enet\_attach()**, there will always be a file open against the Ethernet device, and therefore no channel addresses will be freed even if all the files the application opened are closed. To insure that the channel address will be freed, the ENET\_CLEAR\_CHANNEL **ioctl()** should always be called for an Ethernet file before closing it.

For more information about the **open()** and **close()** functions, refer to the *OS Open Programmer's Reference*.

#### 9.3.2.4 Reading and Writing

After successfully installing and opening the Ethernet port, the **write()** function can be issued. The write buffer must contain a complete Ethernet packet. The universally administered address that was will be copied into the source address field by the device driver. There are prototype Ethernet header structures for both standard Ethernet and 802.3 Ethernet packets in **<enet.h>**. Note that packets must be between 60 and 1514 byte in length (inclusive).

Before reading from the Ethernet file, an additional step must be performed. The Ethernet device driver supports up to 8 receive channels. What this means is that up to 8 files can be open for read or read/write simultaneously, and files will receive only those packets that have been selected for them. Packet selection is by packet type, in the case of standard Ethernet, and by destination SAP in the case of 802.3 Ethernet. The selection address is set with the ioctl ENET\_SET\_CHANNEL command, discussed below.

*fd1* is the value obtained from the **open()** call.

```
fd1 = open("/eno",O_RDWR);
ioctl(fd,ENET_SET_CHANNEL,5,2);
/* send packet from buffer */
write(fd,buffer,count);
/* get received packet into buffer */
read(fd,buffer,count);
close(fd);
```

For more information on **read()** and **write()** functions, refer to the *OS Open Programmer's Reference*.

### 9.3.2.5 I/O Control

The **ioctl()** call issued against the Ethernet device driver accepts the following commands. In each of these commands, *fd* is the value obtained from the **open()** call.

### 9.3.2.6 ENET\_SET\_CHANNEL

This command sets the receive channel address of the file. Once set, a receive channel address cannot be used in a subsequent ioctl ENET\_SET\_CHANNEL command unless it is first cleared with the ioctl ENET\_CLEAR\_CHANNEL command.

```
rc = ioctl(fd, ENET_SET_CHANNEL,
           packet_type,      /* packet type is an unsigned integer containing the channel
                           address */
           type_length); /* specifies how many of the
                           least significant bytes of the
                           packet type are to be used.
                           Only values 1 and 2 are valid. */
```

A word about packet addresses. For standard Ethernet, the packet type is a 2-byte field right after the hardware source address. If *type\_length* is 2, the *packet\_type* parameter is assumed to refer to a standard Ethernet packet type. For a *type\_length* of 1, the *packet\_type* is assumed to contain a 1-byte destination SAP.

The incoming packets are differentiated as follows: For 802.3, there is a length field immediately after the source address. By convention, Ethernet packets are 1500 bytes or less, and valid Ethernet types are > 0x600. Hence, if the field after the source address is less than 0x600, the packet is assumed to be an 802.3 packet, and the 1 byte *packet\_type* is compared against the destination SAP. Some reserved type values should not be generally used. They are defined in the file <netinet/if\_ether.h>.

### 9.3.2.7 ENET\_CLEAR\_CHANNEL

This command clears the receive channel address of the file. This enables the device driver to free up internal resources and return any unread packets on this channel to the receive buffer pool. Once the receive channel address is cleared, it can be used again with the ioctl ENET\_SET\_CHANNEL command. The file can then be set to another receive channel as well.

```
rc = ioctl(fd, ENET_CLEAR_CHANNEL);
```

### 9.3.2.8 ENET\_QUERY\_ADDRESS

This ioctl command retrieves the universally administered address that was assigned during device\_install obtained from the as a part of the device driver initialization

```
unsigned char ua_address[6];
rc = ioctl(fd, ENET_QUERY_ADDRESS, ua_address);
```

The address is copied into the area supplied as the first data parameter to this ioctl.

### 9.3.2.9 Ethernet Device Driver Example

Following is an Ethernet device driver example:

```
#include <stdio.h>
#include <fcntl.h>
#include <sys/devLib.h>
#include <sys/devDriver.h>
#include <sys/ioctl.h>
#include <enet.h>
#define ENET_LENGTH 256
#define PACKET_TYPE 0x7C0
#define ENET_INTLVL 11
#define ENET_REGBASE 0xEA00007
#define ENET_REGDELTA 8
#define ENET_BUFFERS 30
#define ENET_SRAMSZ 8
/* physical Ethernet addresses */
static char target_address[6] = {0x08, 0x00, 0x5A, 0x4D, 0x03, 0x9C};
static char host_address[6] = {0x08, 0x00, 0x5A, 0xCD, 0x00, 0xF5};
/* Install the Ethernet device driver and an Ethernet device, and open an */
/* Ethernet file. Set the receive channel address for the file, build a */
/* frame, and write it out to the predetermined target. */
/* The target will have set up an Ethernet file with the same receive */
/* channel address on its end, and will use read() (possibly in */
/* conjunction with select()) to receive the frame. */
int enet_example(void)
{
    int rc;
    int devhandle;
    int enet_fd;
    int enet_length = ENET_LENGTH;
    int frame_number = 1;
    struct enet_frame *e_frame_ptr;
    rc = driver_install(&devhandle, enet_init, ENET_INTLVL, ENET_REGBASE,
    ENET_BUFFERS, ENET_SRAMSZ, host_address);
    if (rc != 0)
    {
        perror("Driver install failure for Enet");
        return(-1);
    }
    rc = device_install("/dev/en0", CHRTYPE, devhandle);
    if (rc != 0)
    {
```

```

    perror("Device install failure for Enet");
    return(-1);
}
enet_fd=open("/dev/en0", O_RDWR);
if (enet_fd == -1)
{
    perror("Ethernet open failure\n");
    return(-1);
}
rc = ioctl(enet_fd, ENET_SET_CHANNEL, PACKET_TYPE, 2);
if (rc != 0)
{
    perror("Ethernet ENET_SET_CHANNEL ioctl failure\n");
    return -1;
}
printf("\nEthernet file opened with file descriptor %d on channel %x\n",
       enet_fd, PACKET_TYPE);
/* build frame */
e_frame_ptr = (struct enet_frame *) malloc(enet_length);
memcpy(e_frame_ptr->dest_addr,
       target_address,sizeof(target_address));
e_frame_ptr->type = PACKET_TYPE; /* set type */
sprintf(e_frame_ptr->enet_data, \
"Frame number %d from physical addr 0x%2X%2X%2X%2X%2X%2X, \
channel \
0x%X", frame_number, \
host_address[0], host_address[1], host_address[2], \
host_address[3], host_address[4], host_address[5], \
e_frame_ptr->type);
/* write frame */
rc = write(enet_fd, e_frame_ptr, enet_length);
free(e_frame_ptr);
if (rc != enet_length)
{
    perror("Enet write failure");
    return(-1);
}
return(0);
}

```

### 9.3.3 ROM Monitor Ethernet Device Driver

The ROM Monitor Ethernet device driver provides network access to the applications running on the 602 EVB, while still allowing the ROM Monitor to access the RISCWatch debugger over the ethernet.

This device driver uses code resident in the ROM monitor to send and receive ethernet packets. A different IP address must be specified to distinguish the packets from ROM Monitor and OS Open. I/O initialization should be done by calling **dbg\_ioLib\_init()** rather than **ioLib\_init()**.

#### 9.3.3.1 ROM Monitor Ethernet Installation and Initialization

The ROM Monitor Ethernet device driver is installed by calling **biosenet\_attach()**. Following is a prototype of this function:

```
#include <benetLib.h>
int biosenet_attach(unsigned long ipaddr, int init_flag);
```

Upon successful installation, **biosenet\_attach()** returns 0; otherwise -1 is returned. The IP address for OS Open is specified in the *ipaddr* parameter. The *init\_flag* specifies whether the Ethernet controller needs to be initialized. If *init\_flag* is set to 0 then the Ethernet controller is not initialized. If *init\_flag* is set to a non-0 value, initialization of the Ethernet controller is performed.

### 9.3.4 Liquid Crystal Display Device Driver

The Liquid Crystal Display (LCD) device driver controls the optional LCD device. The device driver supports write and I/O control functions only.

#### 9.3.4.1 LCD Device Driver Installation

The LCD device driver is installed by calling **driver\_install()**. Following is an example of device driver installation:

```
#include <lcdLib.h>
int devhandle;
rc=driver_install(&devhandle, lcd_init);
```

**lcd\_init()** is declared in the file **<lcdLib.h>** as follows:

```
int lcd_init(driver_t *dsw, va_list vargs)
```

Upon successful installation, **driver\_install()** returns 0; otherwise -1 is returned. For more information on **driver\_install()**, refer to the *OS Open Programmer's Reference*.

#### 9.3.4.2 LCD Device Installation

After the LCD device driver is installed, a named device can be created using

**device\_install()**. Following is an example of device installation.

```
rc =device_install("/dev/lcd",CHRTYPE,devhandle);
```

For device installation, *devhandle* is the value obtained from the **driver\_install()**. Device type CHRTYPE is defined in **<sys/devDrivr.h>**.

Upon successful installation, **device\_install()** returns 0; otherwise -1 is returned. For more information on **device\_install()**, refer to the *OS Open Programmer's Reference*.

### 9.3.4.3 Opening LCD Device

After the device is installed, the **open()** system call can be used to open the LCD device for writing. Following is an example of the **open()** system call:

```
fd1=open("/dev/lcd", O_WRONLY);
```

### 9.3.4.4 Writing to LCD

After successfully installing and opening the LCD, **write()** calls can be issued against the file descriptor.

Following is an example of a **write()** call:

```
rc=write(fd1,"\\n\\rOS Open Real-time OS\\n", 14);
```

*fd1* is the value obtained from the **open()** call.

The following characters have special meaning when written to the LCD display:

\\r (carriage return) moves the cursor to column 0.

\\n (newline) clears the display, leaving the cursor in the current position

\\b (back space) moves the cursor left one position

If more than sixteen characters are written to the LCD display, an implicit carriage return and line feed are sent to the display causing the display to be cleared and the seventeenth character to be written in the first column.

For more information on **write()**, refer to the *OS Open Programmer's Reference*.

### 9.3.4.5 I/O Control

An **ioctl()** call issued against LCD device driver accepts the commands listed in Table 9-4. All parameter constants can be found in **<lcdLib.h>**

**Table 9-4. ioctl() Commands for the LCD Device Driver**

Command	Parameters	Explanation
LCD_OFF	None	Nothing is displayed
LCD_ON	None	Only characters are displayed

**Table 9-4. ioctl() Commands for the LCD Device Driver**

Command	Parameters	Explanation
LCD_ON_BLINK	None	Characters displayed, cursor character blinks
LCD_ON_CURSOR	None	Characters displayed, cursor displayed
LCD_ON_CURSOR_BLINK	None	Characters displayed, cursor displayed, cursor character blinks
LCD_INIT	None	Initialize display
LCD_CLEAR	None	Blanks, homes and unshifts LCD
LCD_SET_CURSOR	int line, int column	Set cursor to line and column values

Following is an example of an **ioctl()** call issued against LCD device:

```
rc=ioctl(fd1, LCD_CLEAR);
if (rc !=0) printf("ioctl failure\n");
```

*fd1* is the value obtained from the **open()** call.

## 9.4 Environment Bringup and Initialization

The following section describes the processing that occurs when the evaluation board environment is initialized.

Upon power-up or reset the ROM Monitor initializes the processor and other peripherals on the board. If a ROM Monitor load is attempted (via option 0), all enabled power-on tests are executed and, following their completion, a bootp request is sent to the host. This request involves an exchange of UDP packets corresponding to the bootp protocol. In essence, the ROM Monitor asks for and is supplied with the name of the boot image file on the host workstation. **tftp** (Trivial File Transfer Protocol) is then initiated by the ROM Monitor to transfer the boot image to the evaluation board.

Once the file has been transferred, two simple checks are made. A “magic number” in the boot image’s 32-byte header verifies that the image is one that can be loaded by the ROM Monitor (ie., a file created by the eimgbld tool - see appendix B for details of the load format). After the load is complete, control is transferred to the specified entry point in the boot image, which is in the bootstrap program.

When using RISCWatch’s **load image** command to load a boot image file, the debugger strips off the file’s 32-byte header and loads the remaining bytes of the file onto the board.

The start address of the load is designated in bytes 4-8 of the header. Once loaded, the IAR register is set to the boot image's entry point as defined in bytes 16-19 of the header. This entry point is in the bootstrap code. See the "Running Your Programs" section in the RISCWatch User's Guide for additional information on loading files.

#### **9.4.1 Board bootstrap**

The source for OS Open's bootstrap code is included in the **samples\bootLib** directory. The bootstrap program performs the following functions:

1. Unpacks the boot image format, placing the .text and .data sections in the addresses specified at link time.
2. Modifies the kernel configuration block with new heap size and start address.
3. Sets the .bss section to zeros, in accordance with ANSI C requirements.

#### **9.4.2 Environment Initialization**

OS Open requires information about the system environment at initialization. The following source files, which are included with the samples, are used to supply that information and to establish the working environment:

- `basic_os.c` - contains pieces of `config.c`, `io_init.c`, `panic.c`, `thread0.c`, and `utils.c` to provide a minimal OS Open configuration.
- `config.c` - configures the OS Open kernel
- `io_init.c` - initializes OS Open's I/O subsystem
- `network.c` - configures the host names and addresses for your environment
- `panic.c` - provides a sample panic function
- `thread0.c` - configures various features of OS Open (networking, remote debugger, etc.)
- `utils.c` - provides some useful utilities such as `dir()` to produce a directory listing

Additional information can be found in the "Configuring the OS Open Operating System" and "Developing OS Open Applications" chapters in the OS Open User's Guide.

### **9.5 Tools**

Several host tools are provided to assist you in using the EVB support package or creating your own applications for the PowerPC 602. The tools can also be used for ROM program development.

#### **9.5.1 elf2rom**

**elf2rom** takes an ELF format executable file (from the linker/binder), extracts the text and data sections, and writes them to a binary file for use as input to a ROM programmer. This tool can be used by those who wish to modify the ROM Monitor source code and create a new flash memory binary file for use with a ROM programmer or the flash update utility included with EVB software.

## Syntax:

```
elf2rom [-v] [-d] [-p] [-s size] [-i offset] [-o output_file] input_elf
```

## Description:

The program takes the input file *input\_elf*(which is assumed to be an ELF file output from the linker), extracts the text and data sections, and writes them to the file, *output\_file*. There are several optional flags that can affect elf2rom processing. They are described below:

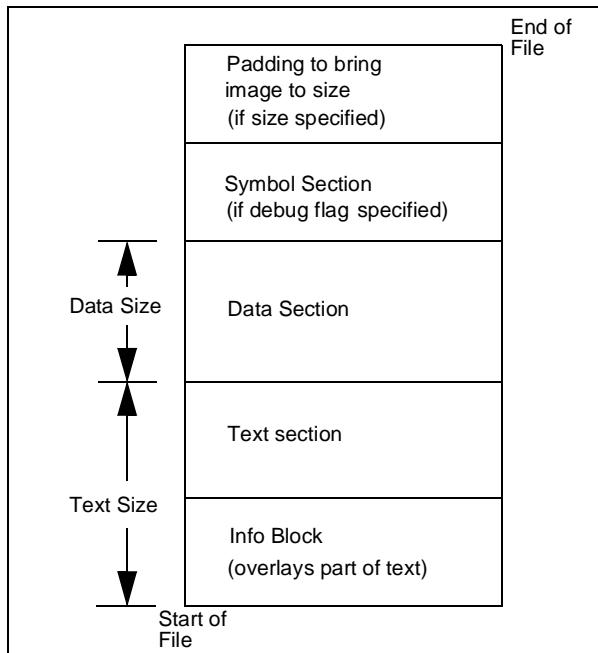
- v                 The verbose flag causes information about the generated output file to be written to stderr at the completion of the utility. This information includes the sizes and origins of the various sections and entry point.
- d                 The debug flag will cause the symbol information from the input ELF file to be included after the data section in the output binary file.
- p                 The promotion flag causes the data section to be aligned on a full word boundary if possible. This alignment facilitates full word moves of data to the appropriate target address without causing alignment exceptions.
- s                 The size flag causes the output binary file to be padded to a particular size. This option is useful if it is necessary to create binary files that are the same size as a target ROM device. Error messages are generated if the generated image exceeds the specified size.
- i offset         The info flag places an information block into the output binary file at the specified offset. Since this info block overlays what is currently in the file at the specified offset, space should be reserved for its placement. The info block contains the following fields:

```
struct info_block {  
    long block_id;         /* Magic Number 0xBFAB0030 */  
    long entry_point;      /* entry point of image */  
    long toc_ptr;          /* used for XCOFF; not used for ELF*/  
    long text_size;        /* size of text section in bytes  
                           also offset from beginning of image to data section */  
    long text_p_addr;      /* text origin address as generated in ELF module */  
    long data_size;        /* size of data section */  
    long data_p_addr;      /* data origin as specified in generated ELF module */  
    long bss_size;         /* size of bss section */  
    long bss_p_addr;       /* bss origin as specified in generated ELF module */  
    long num_syms;        /* number of symbols from symbol section (only  
                           valid if debug flag is set) */  
    long sym_p_addr;       /* address of symbol table. Calculated as text  
                           origin + offset of symbols with created ROM image */  
    long text_offset;      /* offset of text section from beginning of original
```

```
        ELF file. This information is required by  
        certain debuggers */  
};
```

- o output\_file      Allows the specification of an output file name. The default name is input\_elf.img.
- input\_elf            This is simply the ELF binary input file. (elf2rom only)

The following picture shows the relationship of the various sections in the produced output file. The figure assumes that the info block flag [-i] was specified with an offset of 0x00.



**Figure 9-1. elf2rom Output File**

Users can find an example of using elf2rom in the ROM Monitor's Makefile under **osopen/PLATFORM/openbios**.

## 9.5.2 hbranch

**hbranch** places a branch at the end of a ROM image. **hbranch** can also be used to store a communication device's network address in the ROM's Vital Product Data (VPD) area.

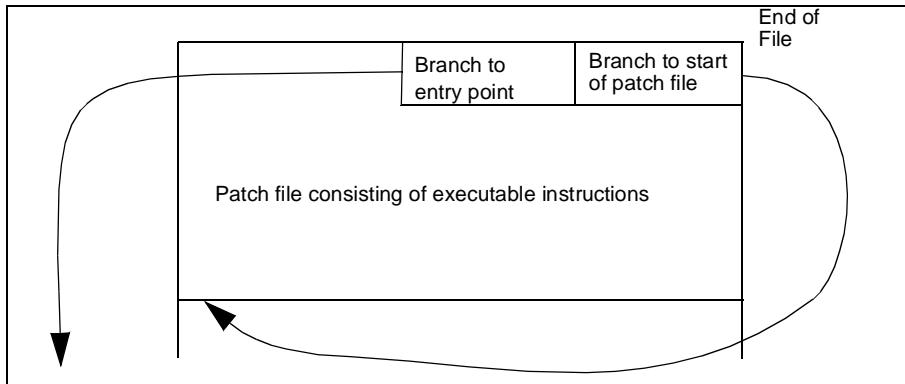
### Syntax:

```
hbranch [-v] [-s size] [-n net_addr] input_image
```

### Description:

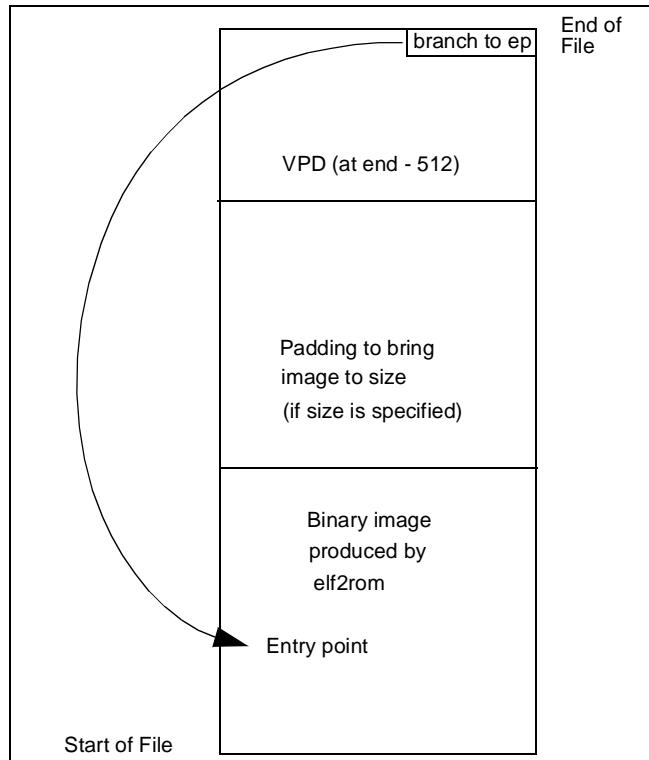
The program takes the input file *input\_image* (which must be the output of `elf2rom`, or `eimgbld` with an information block at 0x0 relative) pads it to size *size* and writes a relative branch to the entry point recorded in the end of the image. The entry point must be a label, not a function descriptor. There are several optional flags that can affect **hbranch** processing. They are described below:

-v	The verbose flag causes information about the generated output image to be written to <i>stderr</i> at the completion of the utility. This information includes entry point information.
-s size	The size flag causes the image to be padded to a particular size. This facility is useful if it is necessary to create binary images that are the same size as a target ROM device.
-n net_addr	The network address flag stores <i>net_addr</i> , a 12 hex character network address (the media access control (MAC) address), in the VPD area in ROM. The ROM Monitor uses this option to store the EVB's ethernet controller's network address in its VPD.
-p patch_file	The patch file flag causes the file <i>patch_file</i> to be placed into the image just before the final branch and logically inserted into the instruction stream between the branch at the end of the file and the entry point. The patch file is inserted into the image "as is" and will usually contain the binary representation of position independent executable instructions. See Figure 9-2 for the details as to how normal hbranch processing is changed by a patch file.
input_image	This is simply the source image file. The output is written to <i>stdout</i> .



**Figure 9-2. Detail of patch file placement**

Figure 9-3 shows the relationship of the various sections in the produced output image.



**Figure 9-3. hbranch Output Image**

Users can find an example of using hbranch in the ROM Monitor's Makefile under **osopen/PLATFORM/openbios**.

### 9.5.3 eimgbld

The **eimgbld** tool converts an output file from the linker/binder into the format used by the ROM Monitor to load programs from the host onto the evaluation board. The ELF file must be an otherwise executable file, with the text and data addresses bound at link time. Since the entry point of the ELF file will be used by the ROM loader, it must point to a suitable bootstrap.

#### Syntax:

```
eimgbld: [ -D -P -S -v -b addr -m m_file -o o_file -s s_file -x x_file] input_elf
```

#### Description:

The program takes the input file *input\_elf* (which must be the final ELF executable file produced from the build process) and converts it into the load format used by the ROM Monitor. There are several optional flags that can affect **eimgbld** processing. They are described below:

- D Set debug flag. A flag is set in the image causing the ROM Monitor debugger to be invoked immediately after the image is loaded.
- P Creates output image in PReP format. PReP format is used by some PowerPC platforms.
- S Suppress symbol information. Specifying this flag will prevent the symbol table from being included in the image.
- v Verbose option. Directs information about the produced image to stderr.
- b *addr* Set the symbol start location to address, *addr*.
- m *m\_file* Specify the ROM address map file. The format of this file is two addresses on each line (start address and ending address separated by a “,”).
- o *o\_file* Allows the specification of an output file name. The default name is *input\_elf.img*.
- s *s\_file* Restrict symbol table to names in specified file, *s\_name*. The format of this file is one symbol on each line.
- x *x\_file* Suppress section names listed in specified file, *x\_name*. The format of this file is one section name on each line.

Users can find an example of using eimgbld in the sample Makefile under **osopen/PLATFORM/samples**.

## 9.5.4 split4

The **split4** program divides a ROM image into four separate files so that each word of the image can be placed across the four 512K flash chips on the 602 EVB. Four binary files are created for use as input to a ROM programmer.

### Syntax:

split4: *input\_image* *out\_name*

### Description:

The program takes the input file *input\_image* and splits each word into four separate bytes and writes them to four binary files for use as input to a ROM programmer. The *input\_image* file must be the output of **elf2rom**. Each byte is written into a separate binary output file depending on their position within the word. For example, the first byte of a word will be written into *out\_name.00*, the second byte will be written into *out\_name.01*, the third byte will be written into *out\_name.10*, and the fourth byte will be written into *out\_name.11*.

<i>input_image</i>	A ROM image that has been generated by <b>elf2rom</b> .
<i>out_name</i>	The file name of the four binary output files. The output files' file types will be 00, 01, 10, and 11.

## 602 EVB Function Reference

---

---

This chapter describes the OS Open functions for the 602 EVB platform. The function calls and macros are arranged alphabetically by name. For information about the effective use of some of these functions, refer to the PowerPC 602 RISC Microprocessor User's Manual.

All descriptions contain the following sections:

- Synopsis
- Library
- Description
- Errors
- Attributes

Examples and references are provided or referenced where appropriate.

### 10.1 Attributes and Threads

Functions and macros have attributes that affect thread execution. Depending on their behavior, functions may or may not be “async safe,” “cancel safe,” and “interrupt handler safe.”

#### 10.1.1 Async Safe Functions

An async safe function may be entered by two or more concurrently executing threads, with each thread getting the correct results.

Functions that operate only on disjoint or local data objects are reentrant, and are therefore async safe. For example, **ppcCntlzw()** operates only on its arguments, making it reentrant and therefore async safe.

Functions that operate on common or global data objects may use serialization techniques, such as mutexes and semaphores, within the functions to ensure async safe operation. **enet\_send\_packet()** uses the functions **semwait()** and **sempost()** to force serialization. Refer to the *OS Open User's Guide* for more information about the use of mutexes and semaphores.

### **10.1.2 Cancel Safe Functions**

The cancel safe attribute is important only to threads executing in deferred cancelability mode (the cancel state is enabled; the cancel type is deferred).

A thread executing in deferred cancelability mode can execute a cancel safe function without being canceled. If the same thread executes a non-cancel safe function, the thread may or may not be canceled during execution of the function.

### **10.1.3 Interrupt Handler Safe Functions**

An interrupt handler safe function may be called by a first level interrupt handler (FLIH).

## **10.2 602 EVB Functions**

Descriptions of the functions provided specifically to support the 602 EVB are listed in Table 10-1:

**Table 10-1. Functions Specific to 602 EVB**

<b>Function or Macro</b>	<b>Description</b>	<b>Page</b>
async_init()	Installs the asynchronous device driver	10-9
biosenet_attach()	Attaches the ROM Monitor Ethernet to an IP address	10-10
clock_set()	Sets the OS Open POSIX clock to the value obtained from the battery operated real time clock	10-12
clockchip_get()	Reads the real-time clock	10-13
clockchip_get_calibration()	Returns the clock's calibration byte	10-14
clockchip_nvram_read()	Reads bytes from the clock chip's NVRAM	10-15
clockchip_nvram_write()	Writes bytes to the clock chip's NVRAM	10-16
clockchip_set_calibration()	Updates the clock's calibration byte with the supplied value	10-18

**Table 10-1. Functions Specific to 602 EVB**

Function or Macro	Description	Page
clockchip_start()	Starts the real-time clock	10-19
clockchip_stop()	Stops the real-time clock	10-20
clockLib_init()	Initializes the clockLib library routines	10-21
dbg_ioLib_init()	Initializes the I/O library	10-22
dcache_flush()	Flushes cache lines, beginning at the effective address and continuing for a specified number of bytes	10-23
dcache_invalidate()	Invalidates cache lines beginning at the effective address and continuing for a specified number of bytes	10-24
ext_int_disable()	Disables the interrupt level specified by an event	10-26
ext_int_enable()	Enables the interrupt level specified by an event	10-27
ext_int_install()	Installs a first level interrupt handler (FLIH) for an event.	10-28
ext_int_query()	Returns information about the FLIH	10-29
ioLib_init()	Initializes I/O library	10-30
lcd_init()	Installs the LCD device driver	10-31
memcpy_io()	Copies memory for memory mapped device drivers	10-32
ppcAbend()	Executes an invalid opcode forcing a program check interrupt	10-33
ppcAndMsr()	ANDs a value with the contents of the MSR	10-34

**Table 10-1. Functions Specific to 602 EVB**

Function or Macro	Description	Page
ppcCtlzw()	Counts consecutive leading zeros in a value	10-35
ppcDcbf()	Copies the cache block back to main storage (if the block resides in cache and has been modified with respect to main storage) and then invalidates the cache block	10-36
ppcDcbi()	Invalidates a cache block, discarding any modified contents if the block is valid in cache	10-37
ppcDcbst()	Copies a cache block, discarding any modified contents if the block is valid in cache	10-38
ppcDcbz()	Sets a cache block to 0	10-39
ppcEieio()	Ensures that all storage references before the call finish before any storage references after the call start	10-40
ppcHalt()	Is a one instruction spin loop, effectively putting the processor in an enabled wait at the point of invocation	10-41
ppclcbi()	Invalidates an instruction cache block	10-42
ppclsync()	Causes the processor to discard any instructions that may have been prefetched	10-43
ppcMfdar()	Returns the value of the processor DAR register	10-44
ppcMfdbat0l() - ppcmfdbat3l()	Returns the value of their respective lower data BAT register.	10-45
ppcMfdbat0u() - ppcmfdbat3u()	Returns the value of their respective upper data BAT register.	10-46
ppcMfdec()	Returns the value of the decrementer.	10-47

**Table 10-1. Functions Specific to 602 EVB**

Function or Macro	Description	Page
ppcMfdsisr()	Returns the value of the data storage interrupt status register (DSISR)	10-48
ppcMfear()	Returns the value of the external access register (EAR)	10-49
ppcMfesasrr()	Returns the value of the ESA Save/Restore Register	10-50
ppcMfgpr1()	Returns the current value of GPR(1)	10-51
ppcMfgpr2()	Returns the current value of GPR(2)	10-52
ppcMfhid0()	Returns the value of the hardware implementation-dependent register 0 (HID0)	10-53
ppcMfhid1()	Returns the value of the hardware implementation-dependent register 1 (HID1)	10-54
ppcMfiabr()	Returns the value of the instruction address breakpoint register (IABR)	10-55
ppcMfibr()	Returns the value of the Interrupt Base Register (IBR)	10-56
ppcMfibat0l() - ppcmfibat3l()	Returns the value of their respective lower instruction BAT register.	10-57
ppcMfibat0u() - ppcmfibat3u()	Returns the value of their respective upper instruction BAT register.	10-58
ppcMflt()	Returns the value of the integer tag register (LT)	10-59
ppcMfmsr()	Returns the value of the MSR	10-60
ppcMfpvr()	Returns the value of the processor version register	10-61

**Table 10-1. Functions Specific to 602 EVB**

Function or Macro	Description	Page
ppcMfsdr1()	Returns the value of storage description register 1	10-62
ppcMfsebr()	Returns the value of the ESA enable base register (SEBR)	10-63
ppcMfser()	Returns the value of the ESA enable register (SER)	10-64
ppcMfsp()	Returns the value of the single-precision tag register (SP)	10-65
ppcMfsprg0()- ppcMfsprg3()	Returns the value of the special purpose register generals (SPRG0 - SPRG3)	10-66
ppcMfsr()	Returns the value of the segment register	10-67
ppcMfsrr0()	Returns the value of SRR0	10-68
ppcMfsrr1()	Returns the current value of SRR1	10-69
ppcMftb()	Returns the current time base data	10-70
ppcMftcr()	Returns the value of the timer control register	10-71
ppcMtdar()	Sets the value of the DAR	10-72
ppcMtdbat0l() - ppcMtdbat3l()	Sets the value of their respective lower data BAT register.	10-73
ppcMtdbat0u() - ppcMtdbat3u()	Sets the value of their respective upper data BAT register.	10-74
ppcMtdec()	Sets the value of the decrementer	10-75
ppcMtdsisr()	Sets the DSISR register	10-76
ppcMtEAR()	Sets the value of the external access register (EAR)	10-77

**Table 10-1. Functions Specific to 602 EVB**

Function or Macro	Description	Page
ppcMtesasrr()	Sets the value of the ESA Save/Restore Register	10-78
ppcMthid0()	Sets the value of the hardware implementation-dependent register 0 (HID0)	10-79
ppcMtibr()	Sets the value of the instruction address breakpoint register (IABR)	10-80
ppcMtibr()	Sets the value of the Interrupt Base Register (IBR)	10-81
ppcMtibat0l() - ppcMtibat3l()	Sets the value of their respective lower instruction BAT register.	10-82
ppcMtibat0u() - ppcMtibat3u()	Sets the value of their respective upper instruction BAT register.	10-83
ppcMtlt()	Sets the value of the integer tag register (LT)	10-84
ppcMtmsr()	Sets the MSR	10-85
ppcMtsdr1()	Sets the SDR1	10-86
ppcMtsebr()	Sets the value of the ESA enable base register (SEBR)	10-87
ppcMtser()	Sets the value of the ESA enable register (SER)	10-88
ppcMtsp()	Sets the value of the single-precision tag register (SP)	10-89
ppcMtsprg0() - ppcMtsprg3()	Sets the special purpose register generals (SPRG0 - SPRG3)	10-90
ppcMtsr()	Sets the segment register	10-91
ppcMtsrr0()	Sets the SRR0	10-92

**Table 10-1. Functions Specific to 602 EVB**

Function or Macro	Description	Page
ppcMtsrr1()	Sets the SRR1	10-93
ppcMtcb()	Sets the value of the current time base	10-94
ppcMtcr()	Sets the timer control register	10-95
ppcOrMsr()	Performs the OR of a value and the current MSR, updating the MSR	10-96
ppcSync()	Causes the processor to wait until all data cache lines scheduled to be written to main storage have actually been written	10-97
s1dbprintf()	A version of <b>printf()</b> that may be used before I/O has been established	10-98
s2dbprintf()	A version of <b>printf()</b> that may be used before I/O has been established for serial port 2	10-99
timertick_install()	Installs and starts the timer tick handler	10-100
timertick_remove()	Removes the timer tick handler	10-101
vs1dbprintf()	A version of <b>printf()</b> that uses polled writes (no interrupts), and may be used before I/O has been established and accepts a va_list as a parameter instead of a variable number of parameters	10-102

## Synopsis

```
#include <sys/asyncLib.h>
int driver_install(int *devhandle,async_init, event_t event);
```

## Library

asyncLib.a

## Description

**asyncLib.a** is the asynchronous device driver that supports the two asynchronous communication ports on the 602 EVB platform. **asyncLib.a** is installed by calling **driver\_install()** with *devhandle* as the first parameter and **async\_init** as the second parameter and *event* as the third parameter.

## Errors

None.

## Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	No

## References

- **driver\_install()** : *OS Open Programmer's Reference*
- “Device Drivers Supplied with the 602 EVB” on page 9-6

# **biosenet\_attach()**

---

## Synopsis

```
#include <benetLib.h>
int biosenet_attach( unsigned long ipaddr, int init_flag);
```

## Library

benetLib.a

## Description

**biosenet\_attach()** attaches the TCP/IP protocol stack to the ROM Monitor Ethernet . Unlike **enet\_init()**, this allows OS Open and the ROM Monitor to share the ethernet device. **biosenet\_attach()** should not be used at the same time as the eneLib.a device driver. The IP address should be different from the IP address defined to the 602 EVB ROM Monitor. *init\_flag* determines if **biosenet\_attach()** should initialize the Ethernet interface. The Ethernet device should be initialized only if OS Open was loaded through an interface other than Ethernet. A non-0 value will cause **biosenet\_attach()** to initialize the Ethernet and a 0 value causes **biosenet\_attach()** not to initialize the Ethernet interface. **biosenet\_attach()** returns 0 if successful and -1 if it is unsuccessful.

**Note:** When using **biosenet\_attach()** the I/O should be initialized by calling **dbg\_ioLib\_init()** rather than **ioLib\_init()**.

**Note:** **biosenet\_attach()** is unavailable for OS Open with Virtual Memory.

## Errors

None.

## Example

Initialize TCP/IP and define an IP address to **biosenet\_attach()**.

```
#include<sys/tcpipLib.h>
int rc;
rc=tcPIP_init("myhostname", 1 , 100);
if (rc!=0) {
    return(-1);
if (net_init() ) return(-1);
return(biosenet_attach(0x07010104,0)); /* specify the IP addr. and the init
flag*/
```

## Attributes

Async Safe

No

Cancel Safe	No
Interrupt Handler Safe	No

### **References**

- “Ethernet Device Driver” on page 9-12
- “ROM Monitor Ethernet Device Driver” on page 9-17

# **clock\_set()**

---

## **Synopsis**

```
#include <clockLib.h>
int clock_set(void);
```

## **Library**

clockLib.a

## **Description**

**clock\_set()** sets the OS Open POSIX clock to the value obtained from the host clock.

## **Errors**

[EIO]	Real-time clock not running.
-------	------------------------------

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **Synopsis**

```
#include <clockLib.h>
int clockchip_get( time_t *timeval );
```

## **Library**

clockLib.a

## **Description**

**clockchip\_get()** reads the battery-backed real-time clock into the *timeval* structure supplied by the user. The clockLib library must be initialized by calling **clockLib\_init()** prior to calling this function.

## **Errors**

[EINVAL]	Library not initialized.
----------	--------------------------

## **Attributes**

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- **clockchip\_set()**, p. 10-17
- **clockLib\_init()**, p. 10-21

# **clockchip\_get\_calibration()**

---

## **Synopsis**

```
#include <clockLib.h>
int clockchip_get_calibration( unsigned char *value );
```

## **Library**

clockLib.a

## **Description**

**clockchip\_get\_calibration()** returns the clock's calibration byte in the variable pointed to by *value*. The calibration byte occupies the five lower order bits of the byte. The sixth bit is a sign bit, "1" indicates positive calibration, "0" indicates negative calibration. The clockLib library must be initialized by calling **clockLib\_init()** prior to calling this function.

## **Errors**

[EINVAL]	Library not initialized.
----------	--------------------------

## **Attributes**

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- **clockchip\_set\_calibration()**, p. 10-18
- **clockLib\_init()**, p. 10-21

## Synopsis

```
#include <clockLib.h>
int clockchip_nvram_read( int index, unsigned char *buffer, int length
 );
```

## Library

clockLib.a

## Description

**clockchip\_nvram\_read()** reads non-volatile RAM from the clock chip. *index* specifies the starting byte of NVRAM, *buffer* points to the location where the bytes will be copied to and *length* specifies the maximum number of bytes to read. **clockchip\_nvram\_read()** returns the actual number of bytes read. The clockLib library must be initialized by calling **clockLib\_init()** prior to calling this function.

**Note:** *index* must be within the range specified during **clockLib\_init()**

## Errors

[EINVAL]	Library not initialized or <i>index</i> out of range.
----------	---

## Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- **clockchip\_nvram\_write()**, p. 10-16
- **clockLib\_init()**, p. 10-21

# **clockchip\_nvram\_write()**

---

## **Synopsis**

```
#include <clockLib.h>
int clockchip_nvram_write( int index, unsigned char *buffer, int
length );
```

## **Library**

clockLib.a

## **Description**

**clockchip\_nvram\_write()** writes non-volatile RAM in the clock chip. *index* specifies the starting byte of NVRAM, *buffer* points to the location where the bytes will be copied from and *length* specifies the maximum number of bytes to write. **clockchip\_nvram\_write()** returns the actual number of bytes written. The **clockLib** library must be initialized by calling **clockLib\_init()** prior to calling this function.

**Note:** *index* must be within the range specified during **clockLib\_init()**

## **Errors**

[EINVAL]	Library not initialized or <i>index</i> out of range.
----------	---

## **Attributes**

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- **clockchip\_nvram\_read()**, p. 10-15
- **clockLib\_init()**, p. 10-21

## Synopsis

```
#include <clockLib.h>
int clockchip_set( time_t timeval );
```

## Library

clockLib.a

## Description

**clockchip\_set()** sets the battery-backed real-time clock to *timeval*, which should contain the number of seconds since January 1st, 1970 UTC.

## Errors

[EIO]	Real-time clock not running.
[EINVAL]	Library not initialized.

## Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- *clock\_set()*, p. 10-12
- *clockLib\_init()*, p. 10-21

# **clockchip\_set\_calibration()**

---

## **Synopsis**

```
#include <clockLib.h>
int clockchip_set_calibration( unsigned char value );
```

## **Library**

clockLib.a

## **Description**

**clockchip\_set\_calibration()** updates the clock's calibration byte with *value*. The calibration byte occupies the five lower order bits of the byte. The sixth bit is a sign bit, “1” indicates positive calibration, “0” indicates negative calibration. Adding bits speeds the clock up and subtracting bits slows the clock down. The clockLib library must be initialized by calling **clockLib\_init()** prior to calling this function.

## **Errors**

[EINVAL]	Library not initialized or too many bits in <i>value</i> byte.
----------	--

## **Attributes**

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- **clockchip\_get\_calibration()**, p. 10-14
- **clockLib\_init()**, p. 10-21

## **Synopsis**

```
#include <clockLib.h>
int clockchip_start( void );
```

## **Library**

clockLib.a

## **Description**

**clockchip\_start()** starts the real-time clock. The clockLib library must be initialized by calling **clockLib\_init()** prior to calling this function.

## **Errors**

[EINVAL]	Library not initialized.
----------	--------------------------

## **Attributes**

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- **clockchip\_stop()**, p. 10-20
- **clockLib\_init()**, p. 10-21

# **clockchip\_stop()**

---

## **Synopsis**

```
#include <clockLib.h>
int clockchip_stop( void );
```

## **Library**

clockLib.a

## **Description**

**clockchip\_stop()** stops the real-time clock. The clockLib library must be initialized by calling **clockLib\_init()** prior to calling this function.

## **Errors**

[EINVAL]	Library not initialized.
----------	--------------------------

## **Attributes**

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- **clockchip\_start()**, p. 10-19
- **clockLib\_init()**, p. 10-21

## Synopsis

```
#include <clockLib.h>
int clockLib_init( unsigned char *regbase, int reg_delta, int
first_index, int last_index);
```

## Library

clockLib.a

## Description

**clockLib\_init()** initializes the clockLib library routines. *regbase* specifies the base address of the clock/nvram chip, *reg\_delta* specifies the distance (in bytes) between each addressable byte in the chip. *first\_index* and *last\_index* indicate the range of bytes in the NVRAM that can be accessed by **clockchip\_nvram\_read()** and **clockchip\_nvram\_write()**. The range is specified using starting and ending index values (inclusive).

**clockLib\_init()** returns 0 if successful.

**Note:** **clockLib\_init()** should be called once at system initialization.

## Errors

[EINVAL]	Already initialized or index out of range.
----------	--

## Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- **clock\_set()**, p. 10-12
- **clockchip\_get()**, p. 10-13
- **clockchip\_get\_calibration()**, p. 10-14
- **clockchip\_nvram\_read()**, p. 10-15
- **clockchip\_nvram\_write()**, p. 10-16
- **clockchip\_set()**, p. 10-17
- **clockchip\_set\_calibration()**, p. 10-18
- **clockchip\_start()**, p. 10-19
- **clockchip\_stop()**, p. 10-20

# **dbg\_ioLib\_init()**

---

## **Synopsis**

```
#include <ioLib.h>
int dbg_ioLib_init( void );
```

## **Library**

ioLib.a

## **Description**

**dbg\_ioLib\_init()** initializes the I/O library. Unlike **ioLib\_init()**, this function allows external I/O interrupts to be screened by the ROM monitor, enabling debug to be performed from outside of the OS Open environment. Only external I/O through IRQ's other than those used by the ROM Monitor are available to OS Open.

If successful, **dbg\_ioLib\_init()** returns 0. Otherwise, **dbg\_ioLib\_init()** returns -1.

## **Errors**

[ENOMEM]	Insufficient memory to allocate first level interrupt handler control areas.
----------	--

## **Attributes**

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

**ioLib\_init()**, p. 10-30

## Synopsis

```
#include <ioLib.h>
void dcache_flush( void *address, unsigned int count );
```

## Library

ioLib.a

## Description

**dcache\_flush()** flushes cache lines, beginning at the effective address and continuing for *count* bytes.

A cache line flush forces the current contents of the cache line to main storage (if the line is valid and marked as modified) and then invalidates the line.

**Note:** Since cache flushes occur on cache line boundaries, the operation can occur outside of the bounds specified by the function call. For example, if *address* is X'216' and *count* is X'12', two cache lines, spanning addresses from X'200' to X'23F', would be flushed.

## Errors

None.

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- dcache\_invalidate(), p. 10-24

# **dcache\_invalidate()**

---

## **Synopsis**

```
#include <ioLib.h >
void dcache_invalidate( void *address, unsigned int count );
```

## **Library**

ioLib.a

## **Description**

**dcache\_invalidate()** invalidates cache lines beginning at the effective address given by *address* and continuing for *count* bytes.

**Note:** Since cache invalidation occurs on cache line boundaries, invalidation can occur outside of the bounds implied by this command. For example, if *address* is X '104' and *count* is 16, the cache line spanning the addresses from X '100' to X '120' would be invalidated.

## **Errors**

None.

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- dcache\_flush(), p. 10-23

## Synopsis

```
#include <enet.h>
int driver_install( int devhandle, enet_init, int enet_intvl, int
register_base, int register_delta, int num_blocks, int sram_size, char
*mac_array);
```

## Library

enetLib.a

## Description

**enetLib.a** is the Ethernet device driver supporting packet level read/writes to the integrated ethernet on the 602 EVB platform. **enetLib.a** is installed by calling **driver\_install()** with eight parameters. The first parameter is the device handle, *devhandle*. The second parameter is the device driver initialization function, **enet\_init**. The third parameter is the ethernet interrupt level, *enet\_intvl*. The fourth parameter is the base I/O address of the Ethernet controller's register set, *register\_base*. The fifth parameter is the number of bytes between the Ethernet controller's registers, *reg\_delta*. The sixth parameter is the number of 256 byte buffers allocated for the Ethernet drivers use, *num\_blocks*. The seventh parameter is the packet sram size, *sram\_size*. The eighth parameter is the location of the universal MAC address assigned to the Ethernet controller, *mac\_array*.

Please see “Ethernet Device Driver” on page 9-12 for additional information.

## Errors

None.

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	No

## References

- **driver\_install()** : OS Open Programmer’s Reference
- “Ethernet Device Driver” on page 9-12

# **ext\_int\_disable()**

---

## **Synopsis**

```
#include <ioLib.h>
void ext_int_disable( int event );
```

## **Library**

ioLib.a

## **Description**

**ext\_int\_disable()** disables the interrupt level specified by *event*. The **ext\_int\_disable()** function returns nothing.

## **Errors**

None.

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- **ext\_int\_enable()**, p. 10-27
- **ext\_int\_install()**, p. 10-28
- **ext\_int\_query()**, p. 10-29
- **ioLib\_init()**, p. 10-30

## **Synopsis**

```
#include <ioLib.h>
void ext_int_enable( int event );
```

## **Library**

ioLib.a

## **Description**

**ext\_int\_enable()** enables the interrupt level specified by *event*.

**ext\_int\_enable()** returns nothing.

## **Errors**

None.

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- **ext\_int\_install()**, p. 10-28
- **ext\_int\_query()**, p. 10-29
- **ioLib\_init()**, p. 10-30

# **ext\_int\_install()**

---

## Synopsis

```
#include <flih.h>
#include <ioLib.h>
int ext_int_install( int event, flih_t *new_flih, flih_t *old_flih );
```

## Library

ioLib.a

## Description

**ext\_int\_install()** installs a first level interrupt handler (FLIH) for *event*.

If *new\_flih* is NULL, the current interrupt handler is removed for the specified event. If *new\_flih* is non-NUL, it points to a **flih\_t** structure containing the following fields:

<i>flih_stack</i>	Pointer to the first stack location; obtained by allocating memory and adding the size of the stack. <i>flih_stack</i> must be 16 byte aligned.
<i>flih_function</i>	Pointer to a function invoked when <i>event</i> occurs.
<i>arg</i>	A user-defined (void *) value passed to <i>flih_function</i> .

If *old\_flih* is not NULL, the previous values of *flih\_function*, *flih\_stack*, and *arg* are stored in the structure pointed to by *old\_flih*.

If successful, **ext\_int\_install()** returns 0. Otherwise, **ext\_int\_install()** returns -1.

## Errors

[EINVAL] *event* does not refer to a valid event.

## Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- **ext\_int\_enable()**, p. 10-27
- **ext\_int\_query()**, p. 10-29
- **ioLib\_init()**, p. 10-30

## Synopsis

```
#include <ioLib.h>
#include <flih.h>
int ext_int_query( int event, flih_t *flih );
```

## Library

ioLib.a

## Description

**ext\_int\_query()** returns information about the first level interrupt handler (FLIH), if any, for *event*.

The *flih* argument points to a **flih\_t** structure containing the following fields:

flih_stack	Pointer to the first stack location; obtained by allocating memory and adding the size of the stack.
flih_function	Pointer to a function invoked when <i>event</i> occurs.
arg	A user-defined ( <code>void *</code> ) value passed to <i>flih_function</i> . If no FLIH is installed for the specified level, each field in the <b>flih_t</b> structure is assigned <code>NULL</code> .

If successful, **ext\_int\_query()** returns 0. Otherwise, **ext\_int\_query()** returns -1.

## Errors

[EINVAL]	<i>event</i> does not refer to a valid event.
----------	---

## Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- **ext\_int\_enable()**, p. 10-27
- **ext\_int\_install()**, p. 10-28
- **ioLib\_init()**, p. 10-30

# ioLib\_init()

---

## Synopsis

```
#include <ioLib.h>
int ioLib_init( void );
```

## Library

ioLib.a

## Description

**ioLib\_init()** initializes the I/O library.

If successful, **ioLib\_init()** returns 0. Otherwise, **ioLib\_init()** returns -1.

**ioLib\_init()** should not be used on a 602 EVB when using the ROM Monitor Ethernet interface or the ROM monitor debugger.

**dbg\_ioLib\_init()** should be used instead.

## Errors

[ENOMEM]	Insufficient memory to allocate first level interrupt handler control areas.
----------	--

## Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## Synopsis

```
#include <lcdLib.h>
int driver_install(int devhandle, lcd_init, int number_lines, int
register_base, int register_delta);
```

## Library

lcdLib.a

## Description

**lcdLib.a** is the LCD device driver. It is installed by calling **driver\_install()** with five parameters. The first parameter is the device handle, *devhandle*. The second parameter is the device driver initialization function, **lcd\_init**. The third parameter is the number of lines contained on the display, *number\_lines*, typically 2. The fourth parameter is the I/O address of the base register, *register\_base*, typically 0x0EB00007. The fifth parameter is the register spacing, *register\_delta*, typically 8. For more information about **lcd\_init** and **driver\_install()**, refer to “Device Drivers Supplied with the 602 EVB” on page 9-6.

## Errors

None.

## Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	No

# **memcpy\_io()**

---

## **Synopsis**

```
#include <ioLib.h>
void * memcpy_io( void *s1, void *s2, size_t len );
```

## **Library**

ioLib.a

## **Description**

**memcpy\_io()** copies the data from *s2* to *s1* for the number of bytes specified by *len*. **memcpy\_io()** is the same as **memcpy()**, but is for use by device drivers which use memory mapped I/O.

**memcpy\_io()** is implemented as a straight byte-at-a-time copy, unlike **memcpy()** which may use a “string” assembler instruction.

**memcpy\_io()** returns the value of *s1*.

## **Errors**

None.

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## Synopsis

```
#include <ppcLib.h>
void ppcAbend(void)
```

## Library

ppcLib.a

## Description

**ppcAbend()** executes an invalid opcode forcing a Program Check interrupt.

## Errors

None.

## Example

- Force an illegal instruction exception:  
`ppcAbend()`

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- *PowerPC 602 RISC Microprocessor User's Manual*

# **ppcAndMsr()**

---

## **Synopsis**

```
#include <ppcLib.h>
unsigned long ppcAndMsr(unsigned long value);
```

## **Library**

ppcLib.a

## **Description**

**ppcAndMsr()** ANDs *value* with the contents of the MSR.

The MSR is updated with the result of the AND operation.

**ppcAndMsr()** returns the previous contents of the MSR.

Refer to the **<ppcLib.h>** file for the defines of the MSR constants:

## **Errors**

None.

## **Example**

- Disable external interrupts.

```
unsigned long orig_msr = ppcAndMsr(~ppcMsrEE);
```

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- **ppcOrMsr()**, p. 10-96
- **ppcMtmsr()**, p. 10-85
- *PowerPC 602 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcCntlzw(unsigned long value);
```

## Library

ppcLib.a

## Description

**ppcCntlzw()** counts consecutive leading zeros in *value*.  
**ppcCntlzw()** returns the count, which ranges from 0 through 32, inclusive.

## Errors

None.

## Example

- Return count of leading zeros in variable k.

```
int k;
unsigned long k = ppcCntlzw(0x0700AA55); /* k = 5 */
```

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- *PowerPC 602 RISC Microprocessor User's Manual*

# **ppcDcbf()**

---

## **Synopsis**

```
#include <ppcLib.h>
void ppcDcbf(void *addr);
```

## **Library**

ppcLib.a

## **Description**

**ppcDcbf()** copies the cache block at the effective address specified by *addr* back to main storage (if the block resides in cache and has been modified with respect to main storage) and then invalidates the cache block.

Effectively, this function acts like **ppcDcbst()** followed by **ppcDcbi()**.

## **Errors**

None.

## **Example**

- Flush the cache line at the effective address X'1000' to main storage and then invalidate the cache line. You might do this in preparation for a DMA slave transfer.

```
ppcDcbf((void *)0x1000);
```

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- ppcDcbst()**, p. 10-38
- ppcDcbi()**, p. 10-37
- ppcDcbz()**, p. 10-39
- PowerPC 602 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
void ppcDcbi(void *addr);
```

## Library

ppcLib.a

## Description

**ppcDcbi()** invalidates the cache block containing *addr*, discarding any modified contents if the block is valid in cache.

## Errors

None.

## Example

- Invalidate the cache line beginning with 0x3000. This might be done before reading an area of storage updated by a DMA transfer.

```
ppcDcbi((void *)0x3000);
```

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- ppcDcbst(), p. 10-38
- ppcDcbi(), p. 10-37
- ppcDcbz(), p. 10-39
- *PowerPC 602 RISC Microprocessor User's Manual*

# **ppcDcbst()**

---

## **Synopsis**

```
#include <ppcLib.h>
void ppcDcbst(void *addr);
```

## **Library**

ppcLib.a

## **Description**

**ppcDcbst()** copies the cache block containing *addr* to main storage, if the block is valid in cache and has been modified with respect to main storage.

## **Errors**

None.

## **Example**

- Force the cache line beginning with 0x4000 to memory if the block is valid and out of sync with storage. This would be done to synchronize the cache and storage without invalidating the cache line.

```
ppcDcbst((void *)0x4000);
```

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- *ppcDcbf()*, p. 10-36
- *ppcDcbi()*, p. 10-37
- *ppcDcbz()*, p. 10-39
- *PowerPC 602 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
void ppcDcbz(void *addr);
```

## Library

ppcLib.a

## Description

**ppcDcbz()** sets the cache block containing the byte referenced by *addr* to 0.

The line is established, if necessary, without fetching the line from main storage.

**Note:** If an invalid real address is specified, problems could occur when a subsequent attempt is made by the cache unit to store that line to main storage.

## Errors

None.

## Example

- Assume buffer is 16 cache lines long and cache aligned. To quickly set it to 0, set to first buffer address.

```
char *bpt = buffer;
for(j = 0; j < 16; j++)
{
    ppcDcbz((void *)bpt);
    bpt += cache_line_size;
}
```

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- ppcDcbf(), p. 10-36
- ppcDcbi(), p. 10-37
- ppcDcbst(), p. 10-38
- PowerPC 602 RISC Microprocessor User's Manual*

# **ppcEieio()**

---

## **Synopsis**

```
#include <ppcLib.h>
void ppcEieio(void);
```

## **Library**

ppcLib.a

## **Description**

**ppcEieio()** ensures that all storage references before the call finish before any storage references after the call start.

## **Errors**

None.

## **Example**

- Ensure storage references are done in order:

```
char *one_loc = (char *)0x202;
char *two_loc = (char *)0x204;

*one_loc = 0xAA; /* write a 0xAA to 0x202 */
ppcEieio(); /* insure the store completes before setting two_loc */
*two_loc = 0x55;
```

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- *PowerPC 602 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
void ppcHalt(void);
```

## Library

ppcLib.a

## Description

**ppcHalt()** is a one instruction spin loop, effectively putting the processor in an enabled wait at the point of invocation.

## Errors

None.

## Example

- Wait at the point of invocation:  
`ppcHalt();`

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- *PowerPC 602 RISC Microprocessor User's Manual*

# **ppclcbi()**

---

## **Synopsis**

```
#include <ppcLib.h>
void ppclcbi(void *addr);
```

## **Library**

ppcLib.a

## **Description**

**ppclcbi()** invalidates the Instruction Cache Block pointed to by the address passed. This may be done after updating an instruction.

## **Errors**

None.

## **Example**

- Write a trap into location 0x3000:

```
unsigned in * i_addr = (int *) 0x3000;
*i_addr = 0x7c800008; /* tw instruction */
ppcDbcst((void *) 0x3000);
ppclcbi((void *) 0x3000);
ppclsync();
```

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- *PowerPC 602 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
void ppclsync(void);
```

## Library

ppcLib.a

## Description

**ppclsync()** causes the processor to discard any instructions that may have been prefetched before **ppclsync()**. This call must be used after modifying instruction storage.

## Errors

None.

## Example

- Place a trap into a given address:

```
*trap_address = 0x7F000008;
ppclsync();
```

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- *PowerPC 602 RISC Microprocessor User's Manual*

# **ppcMfdar()**

---

## **Synopsis**

```
#include <ppcLib.h>
unsigned long ppcMfdar(void);
```

## **Library**

ppcLib.a

## **Description**

**ppcMfdar()** returns the value of the processor DAR register.

When a Data Storage interrupt occurs, indicating a data storage access failed (this can happen for a variety of reasons) the DAR is set to the data address triggering the failure.

## **Errors**

None.

## **Example**

- Retrieve the value of DAR register. An alignment exception handler would require the value of the DAR.

```
unsigned long current_DAR=ppcMfdar();
```

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- *PowerPC 602 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfdbat0l(void)
unsigned long ppcMfdbat1l(void)
unsigned long ppcMfdbat2l(void)
unsigned long ppcMfdbat3l(void)
```

## Library

ppcLib.a

## Description

**ppcMfdbat0l() - ppcMfdbat3l()** returns the current value of the specified lower Data Block Address Translation Register (DBAT). The DBAT maintains the address translation information for a block of memory. Each lower DBAT is paired with an upper DBAT register.

## Errors

None.

## Example

- Retrieve the current value of the DBAT3L register:  
`unsigned long dbat3l_value= ppcMfdbat3l();`

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## Processors

PowerPC 602	Yes
PowerPC 603	Yes
PowerPC 604	Yes

## References

- *ppcMfdbat0u() - ppcMfdbat3u(), p. 10-46*
- *PowerPC 602 RISC Microprocessor User's Manual*

# **ppcMfdbat0u() - ppcMfdbat3u()**

---

## **Synopsis**

```
#include <ppcLib.h>
unsigned long ppcMfdbat0u(void)
unsigned long ppcMfdbat1u(void)
unsigned long ppcMfdbat2u(void)
unsigned long ppcMfdbat3u(void)
```

## **Library**

ppcLib.a

## **Description**

**ppcMfdbat0u() - ppcMfdbat3u()** returns the current value of the specified upper Data Block Address Translation Register (DBAT). The DBAT maintains the address translation information for a block of memory. Each upper DBAT is paired with a lower DBAT register.

## **Errors**

None.

## **Example**

- Retrieve the current value of the DBAT3U register:  
`unsigned long dbat3ur_value= ppcMfdbat3u();`

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **Processors**

PowerPC 602	Yes
-------------	-----

## **References**

- *ppcMfdbat0l() - ppcMfdbat3l(), p. 10-45*
- *PowerPC 602 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfdec(void);
```

## Library

ppcLib.a

## Description

**ppcMfdec()** returns the value of the decrementer.

The decrementer is a continuously running 32 bit binary down counter that can assert an interrupt request whenever bit 0 (MSB) changes from 0 to 1. Although not all of the least significant bits may be implemented, bit 31 effectively changes every nanosecond.

## Errors

None.

## Example

- Retrieve the current value of the decrementer. An exception handler for the decrementer may require the value to adjust the value of the next “tick”.

```
unsigned long current_dec=ppcMfdec();
```

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- *PowerPC 602 RISC Microprocessor User’s Manual*

# **ppcMfdsisr()**

---

## **Synopsis**

```
#include <ppcLib.h>
unsigned long ppcMfdsisr(void);
```

## **Library**

ppcLib.a

## **Description**

**ppcMfdsisr()** returns the value of the Data Storage Interrupt Status Register (DSISR) register.

This register indicates the reason for a Data Storage Interrupt.

## **Errors**

None.

## **Example**

- Retrieve the current value of the data storage interrupt status register. An alignment exception handler could take advantage of the partial instruction decoding provided by this register.

```
unsigned long current_dsisr=ppcMfdsisr();
```

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- *PowerPC 602 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfear(void);
```

## Library

ppcLib.a

## Description

**ppcMfear()** returns the value of the External Access Register (EAR). This register identifies the target device for external control operations.

## Errors

None.

## Example

- Retrieve the current value of the EAR.

```
unsigned long current_ear=ppcMfear();
```

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- *PowerPC 602 RISC Microprocessor User's Manual*
- *PowerPC Microprocessor Family: The Programming Environments*

# **ppcMfesasrr()**

---

## **Synopsis**

```
#include <ppcLib.h>
unsigned long ppcMfesasrr(void);
```

## **Library**

ppcLib.a

## **Description**

**ppcMfesasrr()** returns the value of the ESA Save/Restore Register (ESASRR).

## **Errors**

None.

## **Example**

- Retrieve the current value of the ESASRR.

```
unsigned long current_esasrr=ppcMfesasrr();
```

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- *PowerPC 602 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfgpr1(void);
```

## Library

ppcLib.a

## Description

**ppcMfgpr1()** returns the current value of GPR(1).

Typically, this is the value of the current stack frame.

## Errors

None.

## Example

See **ppcMfgpr2()**, p. 10-52.

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
PowerPC 602	Yes

## References

- *PowerPC 602 RISC Microprocessor User's Manual*

# **ppcMfgpr2()**

---

## **Synopsis**

```
#include <ppcLib.h>
unsigned long ppcMfgpr2(void)
```

## **Library**

ppcLib.a

## **Description**

**ppcMfgpr2()** returns the current value of GPR(2).

For XCOFF-based OS Open this is typically the value of the table of contents (TOC) pointer for the current execution context.

## **Errors**

None.

## **Example**

- Retrieve TOC and stack frame base from current context:

```
toc = ppcMfgpr2();
unsigned long stack_base = ppcMfgpr1();
```

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- *PowerPC 602 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfhid0(void)
```

## Library

ppcLib.a

## Description

**ppcMfhid0()** returns the current value of the Hardware Implementation Dependent Register 0 (HID0).

## Errors

None.

## Example

- Retrieve HID0 value:

```
unsigned long hid0_value= ppcMfhid0();
```

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- *PowerPC 602 RISC Microprocessor User's Manual*

# **ppcMfhid1()**

---

## **Synopsis**

```
#include <ppcLib.h>
unsigned long ppcMfhid1(void)
```

## **Library**

ppcLib.a

## **Description**

**ppcMfhid1()** returns the current value of the Hardware Implementation Dependent Register 1(HID1).

## **Errors**

None.

## **Example**

- Retrieve HID1 value:  

```
unsigned long hid1_value= ppcMfhid1();
```

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- *PowerPC 602 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfiabr(void)
```

## Library

ppcLib.a

## Description

**ppcMfiabr()** returns the current value of the Instruction Address Breakpoint Register(IABR). When the IABR is enabled, the instruction fetch address will be compared with the effective address that is stored in the IABR. The compares will be done on a word boundary. If the IABR and the instruction fetch address compare the instruction breakpoint handler will be invoked.

## Errors

None.

## Example

- Retrieve IABR value:

```
unsigned long iabr_value= ppcMfiabr();
```

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- *PowerPC 602 RISC Microprocessor User's Manual*

# **ppcMfibr()**

---

## **Synopsis**

```
#include <ppcLib.h>
unsigned long ppcMfibr(void)
```

## **Library**

ppcLib.a

## **Description**

**ppcMfibr()** returns the current value of the Interrupt Base Register (IBR).

## **Errors**

None.

## **Example**

- Retrieve IBR value:

```
unsigned long ibr_value = ppcMfibr();
```

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- *PowerPC 602 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfibat0l(void)
unsigned long ppcMfibat1l(void)
unsigned long ppcMfibat2l(void)
unsigned long ppcMfibat3l(void)
```

## Library

ppcLib.a

## Description

**ppcMfibat0l() - ppcMfibat3l()** returns the current value of the specified lower Instruction Block Address Translation Register (IBAT). The IBAT maintains the address translation information for a block of memory. Each lower IBAT is paired with an upper IBAT register.

## Errors

None.

## Example

- Retrieve the current value of the IBAT3Lregister:  
`unsigned long ibat3l_value= ppcMfibat3l();`

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- *ppcMfibat0u() - ppcMfibat3u(), p. 10-58*
- *PowerPC 602 RISC Microprocessor User's Manual*
- *PowerPC Microprocessor Family: The Programming Environments*

# **ppcMfibat0u() - ppcMfibat3u()**

---

## **Synopsis**

```
#include <ppcLib.h>
unsigned long ppcMfibat0u(void)
unsigned long ppcMfibat1u(void)
unsigned long ppcMfibat2u(void)
unsigned long ppcMfibat3u(void)
```

## **Library**

ppcLib.a

## **Description**

**ppcMfibat0u() - ppcMfibat3u()** returns the current value of the specified upper Instruction Block Address Translation Register (IBAT). The IBAT maintains the address translation information for a block of memory. Each upper IBAT is paired with a lower IBAT register.

## **Errors**

None.

## **Example**

- Retrieve the current value of the IBAT3U register:

```
unsigned long ibat3u_value= ppcMfibat3u();
```

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- *ppcMfibat0l() - ppcMfibat3l()*, p. 10-57
- *PowerPC 602 RISC Microprocessor User's Manual*
- *PowerPC Microprocessor Family: The Programming Environments*

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMflt(void)
```

## Library

ppcLib.a

## Description

**ppcMflt()** returns the current value of the Integer Tag Register (LT).

## Errors

None.

## Example

- Retrieve LT value:

```
unsigned long lt_value = ppcMflt();
```

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- *PowerPC 602 RISC Microprocessor User's Manual*

# **ppcMfmsr()**

---

## **Synopsis**

```
#include <ppcLib.h>
unsigned long ppcMfmsr(void);
```

## **Library**

ppcLib.a

## **Description**

**ppcMfmsr()** returns the value of the Machine State Register(MSR).

Refer to the [\*\*<ppc\\_arch.h>\*\*](#) file for the defines of constants that can be used as masks with the MSR value.

## **Errors**

None.

## **Example**

See **ppcMtmsr()**, p. 10-85.

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- *PowerPC 602 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfpvr(void);
```

## Library

ppcLib.a

## Description

**ppcMfpvr()** returns the value of the processor version register, which indicates the version and revision of the PowerPC processor.

## Errors

None.

## Example

- Retrieve the current value of the processor version register. Processor version-specific code may require this value:

```
printf("This is processor version %x\n", ppcMfpvr());
```

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- *PowerPC 602 RISC Microprocessor User's Manual*

# **ppcMfsdr1()**

---

## **Synopsis**

```
#include <ppcLib.h>
unsigned long ppcMfsdr1(void);
```

## **Library**

ppcLib.a

## **Description**

**ppcMfsdr1()** returns the value of Storage Description Register 1, which contains high order address bits of the Page Frame Table used in address translation and the hash mask.

## **Errors**

None.

## **Example**

- Retrieve the value of the storage description register 1. This register provides the anchor for structures used to manage virtual storage mapping for the PowerPC 6xx processors.

```
unsigned long current_sdr1=ppcMfsdr1();
```

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- *PowerPC 602 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfsebr(void)
```

## Library

ppcLib.a

## Description

**ppcMfsebr()** returns the current value of the ESA Enable Base Register (SEBR).

## Errors

None.

## Example

- Retrieve SEBR value:

```
unsigned long sebr_value = ppcMfsebr();
```

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- *PowerPC 602 RISC Microprocessor User's Manual*

# **ppcMfser()**

---

## **Synopsis**

```
#include <ppcLib.h>
unsigned long ppcMfser(void)
```

## **Library**

ppcLib.a

## **Description**

**ppcMfser()** returns the current value of the ESA Enable Register (SER).

## **Errors**

None.

## **Example**

- Retrieve SER value:

```
unsigned long ser_value = ppcMfser();
```

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- *PowerPC 602 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfsp(void)
```

## Library

ppcLib.a

## Description

**ppcMfsp()** returns the current value of the Single-Precision Tag Register (SP).

## Errors

None.

## Example

- Retrieve SP value:

```
unsigned long sp_value = ppcMfsp();
```

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- *PowerPC 602 RISC Microprocessor User's Manual*

# **ppcMfsprg0() - ppcMfsprg3()**

---

## **Synopsis**

```
#include <ppcLib.h>
unsigned long ppcMfsprg0(void);
unsigned long ppcMfsprg1(void);
unsigned long ppcMfsprg2(void);
unsigned long ppcMfsprg3(void);
```

## **Library**

ppcLib.a

## **Description**

**ppcMfsprg0() - ppcMfsprg3()** returns the current value of the special purpose register generals (SPRG0 - SPRG3).

Typically, the SPRGs provide temporary storage at the operating system level.

**NOTE:** OS Open reserves these registers for its own use.

## **Errors**

None.

## **Example**

- Read value of SPRG0:

```
unsigned long sprg0_value = ppcMfsprg0();
```

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- *PowerPC 602 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfsr(unsigned long seg_offset);
```

## Library

ppcLib.a

## Description

**ppcMfsr()** returns the value of the segment register given by *seg\_offset*. Valid values for *seg\_offset* are 0-15 inclusive.

## Errors

None.

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- *PowerPC 602 RISC Microprocessor User's Manual*

# **ppcMfsrr0()**

---

## **Synopsis**

```
#include <ppcLib.h>
unsigned long ppcMfsrr0(void);
```

## **Library**

ppcLib.a

## **Description**

**ppcMfsrr0()** returns the value of SRR0.

Typically, SRR0 is used in interrupt handlers, as it usually contains the address of the next instruction to be executed at the time of the interrupt.

## **Errors**

None.

## **Example**

- Retrieve the current value of the SRR0. An exception handler may use this value to determine the point of exception.

```
unsigned long current_srr0=ppcMfsrr0();
```

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- **ppcMfsrr1()**, p. 10-69
- *PowerPC 602 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfsrr1(void);
```

## Library

ppcLib.a

## Description

**ppcMfsrr1()** returns the current value of SRR1.

Typically, SRR1 is used in interrupt handlers, as it contains the old MSR value as well as information bits specific to the interrupt.

## Errors

None.

## Example

- Retrieve the current value of SRR1. This register contains the saved MSR, which may be needed by an exception handler.

```
unsigned long current_srr1=ppcMfsrr1();
```

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- *PowerPC 602 RISC Microprocessor User's Manual*

# **ppcMftb()**

---

## **Synopsis**

```
#include <ppcLib.h>
void ppcMftb(tb_t *clock_data);
```

## **Library**

ppcLib.a

## **Description**

**ppcMftb()** returns the current time base data.

Typically, the time base registers are used to determine the number of clock cycles that have passed.

## **Errors**

None.

## **Example**

- Retrieve the current value of time base high and low registers:

```
tb_t clock_data;
ppcMftb(&clock_data);
```

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- *PowerPC 602 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMftcr(void);
```

## Library

ppcLib.a

## Description

**ppcMftcr()** returns the value of the Timer Control Register.

## Errors

None.

## Example

- Retrieve the current value of TCR register:

```
unsigned long tcr_value;
tcr_value = ppcMftcr();
```

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- *PowerPC 602 RISC Microprocessor User's Manual*

# **ppcMtdar()**

---

## **Synopsis**

```
#include <ppcLib.h>
void ppcMtdar(unsigned long dar_value);
```

## **Library**

ppcLib.a

## **Description**

**ppcMtdar()** sets the value of the DAR to *dar\_value*.

## **Errors**

None.

## **Example**

- Set the data address register to 0x1234. It may be necessary to simulate an exception condition to subsequent software.  
`ppcMtdar(0x1234);`

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- *PowerPC 602 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMtdbat0l(void)
unsigned long ppcMtdbat1l(void)
unsigned long ppcMtdbat2l(void)
unsigned long ppcMtdbat3l(void)
```

## Library

ppcLib.a

## Description

**ppcMtdbat0l() - ppcMtdbat3l()** sets the current value of the specified lower Data Block Address Translation Register (DBAT). The DBAT maintains the address translation information for a block of memory. Each lower DBAT is paired with an upper DBAT register.

## Errors

None.

## Example

- Retrieve the current value of the DBAT3L register and set to inhibit caching.

```
unsigned long dbat3l_value= ppcMfdbat3l();
ppcMtdbat3l(dbat3l_value | 0x00000020);
```

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- *ppcMtdbat0u() - ppcMtdbat3u(), p. 10-74*
- *PowerPC 602 RISC Microprocessor User's Manual*
- *PowerPC Microprocessor Family: The Programming Environments*

# **ppcMtdbat0u() - ppcMtdbat3u()**

---

## **Synopsis**

```
#include <ppcLib.h>
unsigned long ppcMtdbat0u(void)
unsigned long ppcMtdbat1u(void)
unsigned long ppcMtdbat2u(void)
unsigned long ppcMtdbat3u(void)
```

## **Library**

ppcLib.a

## **Description**

**ppcMtdbat0u() - ppcMtdbat3u()** sets the value of the specified upper Data Block Address Translation Register (DBAT). The DBAT maintains the address translation information for a block of memory. Each upper DBAT is paired with a lower DBAT register.

## **Errors**

None.

## **Example**

- Retrieve the current value of the DBAT3U register and update the block length to 1 Mbytes.

```
unsigned long dbat3u_value= ppcMfdbat3u();
ppcMtdbat3u(dbat3u_value & 0xffffe01f);
```

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- *ppcMtdbat0l() - ppcMtdbat3l(), p. 10-73*
- *PowerPC 602 RISC Microprocessor User's Manual*
- *PowerPC Microprocessor Family: The programming Environments*

## Synopsis

```
#include <ppcLib.h>
void ppcMtdec(unsigned long dec_value);
```

## Library

ppcLib.a

## Description

**ppcMtdec()** sets the value of the decrementer to *dec\_value*.

The decrementer is a continuously running 32-bit binary down counter that can assert an interrupt request whenever bit 0 (MSB) changes from 0 to 1. Although not all of the least significant bits may be implemented, bit 31 effectively changes every nanosecond.

## Errors

None.

## Example

- Set the decrementer register to X'0'. A decrementer exception handler may need to reset the value of the decrementer to a specific value.

```
ppcMtdec(0x0);
```

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- *PowerPC 602 RISC Microprocessor User's Manual*

# **ppcMtdsisr()**

---

## **Synopsis**

```
#include <ppcLib.h>
void ppcMtdsisr(unsigned long dsisr_value);
```

## **Library**

ppcLib.a

## **Description**

**ppcMtdsisr()** sets the DSISR register to *dsisr\_value*.

## **Errors**

None.

## **Example**

- Set the data storage interrupt status register to X'00004321'.  
ppcMtdsisr(0x00004321);

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- *PowerPC 602 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
void ppcMtear(unsigned long ear_value);
```

## Library

ppcLib.a

## Description

**ppcMtear()** set the External Access Register (EAR) to the specified value. This register identifies the target device for external control operations. Bit 0 is the enable bit for **eciwx** and **ecowx** instructions. Bits 1-25 are reserved. Bits 26-31 contain the resource ID.

## Errors

None.

## Example

- Set the value of the EAR to enable **eciwx** and **ecowx** instructions for target device 0xa.

```
ppcMtear(0x1000000a);
```

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- PowerPC 602 RISC Microprocessor User's Manual*
- PowerPC Microprocessor Family: The Programming Environments*

# **ppcMtesasrr()**

---

## **Synopsis**

```
#include <ppcLib.h>
void ppcMtesasrr(unsigned long ear_value);
```

## **Library**

ppcLib.a

## **Description**

**ppcMtesasrr()** sets the ESA Save/Restore Register (ESASRR) to the specified value.

## **Errors**

None.

## **Example**

- Set the value of the ESASRR 0x00000001 (EE).  
ppcMtesasrr(0x00000001);

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- *PowerPC 602 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
void ppcMthid0(unsigned long hid0_value)
```

## Library

ppcLib.a

## Description

**ppcMthid0()** set the value of the Hardware Implementation Dependent Register 0 (HID0).

## Errors

None.

## Example

- Set the Instruction Cache Enable bit in HID0:

```
unsigned long hid0_value= ppcMfhid0();
ppcMthid0(hid0_value|0x00008000);
```

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- *PowerPC 602 RISC Microprocessor User's Manual*

# **ppcMtiabr()**

---

## **Synopsis**

```
#include <ppcLib.h>
void ppcMfiabr(iunsigned long)
```

## **Library**

ppcLib.a

## **Description**

**ppcMtiabr()** sets the specified value of the Instruction Address Breakpoint Register(IABR). When the IABR is enabled, the instruction fetch address will be compared with the effective address that is stored in the IABR. The compare will be done on a word boundary. If the IABR and the instruction fetch address match, the instruction address breakpoint exception will be taken. Bits 0-29 is the address of the requested instruction. Bit 30 is the breakpoint enable bit. Bit 31 is reserved for other processors.

## **Errors**

None.

## **Example**

- Set the IABR to invoke the instruction breakpoint handler before the instruction at the real address of 0x0000a004 is executed.  
ppcMtiabr(0x0000a006);

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- *PowerPC 602 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
void ppcMtibr(unsigned long ear_value);
```

## Library

ppcLib.a

## Description

**ppcMtibr()** sets the Interrupt Base Register (IBR) to the specified value.

## Errors

None.

## Example

- Set the value of the IBR 0x00020000.

```
ppcMtibr(0x00020000);
```

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- *PowerPC 602 RISC Microprocessor User's Manual*

# **ppcMtibat0l() - ppcMtibat3l()**

---

## **Synopsis**

```
#include <ppcLib.h>
unsigned long ppcMtibat0l(void)
unsigned long ppcMtibat1l(void)
unsigned long ppcMtibat2l(void)
unsigned long ppcMtibat3l(void)
```

## **Library**

ppcLib.a

## **Description**

**ppcMtibat0l() - ppcMtibat3l()** sets the current value of the specified lower Instruction Block Address Translation Register (IBAT). The IBAT maintains the address translation information for a block of memory. Each lower IBAT is paired with an upper IBAT register.

## **Errors**

None.

## **Example**

- Retrieve the current value of the IBAT3L register and set to inhibit caching.

```
unsigned long ibat3l_value= ppcMfibat3l();
ppcMtibat3l(ibat3l_value | 0x00000020);
```

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- *ppcMtdbat0u() - ppcMtdbat3u()*, p. 10-74
- *PowerPC 602 RISC Microprocessor User's Manual*
- *PowerPC Microprocessor Family: The programming Environments*

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMtibat0u(void)
unsigned long ppcMtibat1u(void)
unsigned long ppcMtibat2u(void)
unsigned long ppcMtibat3u(void)
```

## Library

ppcLib.a

## Description

**ppcMtibat0u() - ppcMtibat3u()** sets the value of the specified upper Instruction Block Address Translation Register (IBAT). The IBAT maintains the address translation information for a block of memory. Each upper IBAT is paired with a lower IBAT register.

## Errors

None.

## Example

- Retrieve the current value of the IBAT3U register and update the block length to 1 Mbytes.

```
unsigned long ibat3u_value= ppcMfibat3u();
ppcMtibat3u(ibat3u_value & 0xffffe01f);
```

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- *ppcMtibat0l() - ppcMtibat3l(), p. 10-82*
- *PowerPC 602 RISC Microprocessor User's Manual*
- *PowerPC Microprocessor Family: The programming Environments*

# **ppcMtlt()**

---

## **Synopsis**

```
#include <ppcLib.h>
void ppcMtlt(unsigned long ear_value);
```

## **Library**

ppcLib.a

## **Description**

**ppcMtlt()** sets the Integer Tag Register (LT) to the specified value.

## **Errors**

None.

## **Example**

- Set the value of the LT 0x00000001.

```
ppcMtlt(0x00000001);
```

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- *PowerPC 602 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
void ppcMtmsr(unsigned long msr_value);
```

## Library

ppcLib.a

## Description

**ppcMtmsr()** sets the MSR to *msr\_value*.

The file **<ppc\_arch.h>** defines constants that can be used with the MSR:

## Errors

None.

## Example

- Enable external interrupts:

```
unsigned long msr = ppcMfmsr();
ppcMtmsr(msr | ppcMsrEE);
```

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- *PowerPC 602 RISC Microprocessor User's Manual*

# **ppcMtsdr1()**

---

## **Synopsis**

```
#include <ppcLib.h>
void ppcMtsdr1(unsigned long sdr1_value);
```

## **Library**

ppcLib.a

## **Description**

**ppcMtsdr1()** sets the SDR1 to *sdr1\_value*.

## **Errors**

None.

## **Example**

- Set the storage description register 1 to X'A0000'. Setting the SDR1 register would usually be done by software managing virtual translation.  
`ppcMtsdr1(0xA0000);`

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- *PowerPC 602 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
void ppcMtsebr(unsigned long ear_value);
```

## Library

ppcLib.a

## Description

**ppcMtsebr()** sets the ESA Enable Base Register (SEBR) to the specified value.

## Errors

None.

## Example

- Set the value of the SEBR 0x00010000.  
`ppcMtsebr(0x00010000);`

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- *PowerPC 602 RISC Microprocessor User's Manual*

# **ppcMtser()**

---

## **Synopsis**

```
#include <ppcLib.h>
void ppcMtser(unsigned long ear_value);
```

## **Library**

ppcLib.a

## **Description**

**ppcMtser()** sets the ESA Enable Register (SER) to the specified value.

## **Errors**

None.

## **Example**

- Set the value of the SER 0x00000001.

```
ppcMtser(0x00000001);
```

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- *PowerPC 602 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
void ppcMtsp(unsigned long ear_value);
```

## Library

ppcLib.a

## Description

**ppcMtsp()** sets the Single-Precision Tag Register (SP) to the specified value.

## Errors

None.

## Example

- Set the value of the SP 0x00000001.  
ppcMtsp(0x00000001);

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## Processors

PowerPC 602	Yes
PowerPC 603	No

## References

- *PowerPC 602 RISC Microprocessor User's Manual*

# **ppcMtsprg0() - ppcMtsprg3()**

---

## **Synopsis**

```
#include <ppcLib.h>
void ppcMtsprg0(unsigned long data);
void ppcMtsprg1(unsigned long data);
void ppcMtsprg2(unsigned long data);
void ppcMtsprg3(unsigned long data);
```

## **Library**

ppcLib.a

## **Description**

**ppcMtsprg0() - ppcMtsprg3()** set the special purpose register generals (SPRG0 - SPRG3) to the specified values.

Typically, the SPRGs provide temporary storage at the operating system level.

**NOTE:** OS Open reserves these registers for its own use.

## **Errors**

None.

## **Example**

- Set SPRG0 to 0xA0000000:

```
ppcMtsprg0(0xA0000000);
```

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- *PowerPC 602 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
void ppcMtsr(unsigned long seg_offset, unsigned long value);
```

## Library

ppcLib.a

## Description

**ppcMtsr()** sets the segment register given by *seg\_offset* to *value*.

## Errors

None.

## Example

- Set segment register 7 to memory forced I/O.

```
ppcMtsr(7, 0x87F00007);
```

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- *PowerPC 602 RISC Microprocessor User's Manual*

# **ppcMtsrr0()**

---

## **Synopsis**

```
#include <ppcLib.h>
void ppcMtsrr0(unsigned long srr0_value);
```

## **Library**

ppcLib.a

## **Description**

**ppcMtsrr0()** sets the SRR0 to *srr0\_value*.

## **Errors**

None.

## **Example**

- Set the save/restore register 0 to X'DF000000':

```
ppcMtsrr0(0xDF000000);
```

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- *PowerPC 602 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
void ppcMtsrr1(unsigned long srr1_value);
```

## Library

ppcLib.a

## Description

**ppcMtsrr1()** sets the SRR1 to *srr1\_value*.

## Errors

None.

## Example

- Set the save/restore register 1 to X'0000BB00':

```
ppcMtsrr1(0x0000BB00);
```

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- *PowerPC 602 RISC Microprocessor User's Manual*

# **ppcMttb()**

---

## **Synopsis**

```
#include <ppcLib.h>
void ppcMttb(tb_t *clock_data);
```

## **Library**

ppcLib.a

## **Description**

**ppcMttb()** sets the current time base data.

Typically, the time base registers are used to determine the number of clock cycles that have passed.

## **Errors**

None.

## **Example**

- Set the current value of time base high and low registers:

```
tb_t clock_data;
ppcMttb(0x00000000);
```

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- *PowerPC 602 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
void ppcMttcr(unsigned long tcr_value);
```

## Library

ppcLib.a

## Description

**ppcMttcr()** sets the timer control register to the specified value.

## Errors

None.

## Example

- Set the TCR to enable watchdog timer interrupts:

```
ppcMttcr(0x04000000);
```

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- *PowerPC 602 RISC Microprocessor User's Manual*

# **ppcOrMsr()**

---

## **Synopsis**

```
#include <ppcLib.h>
unsigned long ppcOrMsr(unsigned long value);
```

## **Library**

ppcLib.a

## **Description**

**ppcOrMsr()** performs the OR of *value* and the current MSR, updating the MSR.

The previous value of the MSR is returned.

The file **<ppcLib.h>** defines several constants for the MSR that can be used as masks:

## **Errors**

None.

## **Example**

- Enable instruction address translation.

```
unsigned long old_val = ppcOrMsr(ppcMsrIR);
```

## **Attributes**

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- *PowerPC 602 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
void ppcSync(void);
```

## Library

ppcLib.a

## Description

**ppcSync()** causes the processor to wait until all data cache lines scheduled to be written to main storage have actually been written.

## Errors

None.

## Example

- Ensure a **ppcDbci()** completes before using the values:

```
char *memptr = (char *)0x2000;
char new_value;
ppcDbci((void *)memptr)
ppcSync();
new_value = *memptr;
```

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- *PowerPC 602 RISC Microprocessor User's Manual*

# s1dbprintf()

---

## Synopsis

```
#include <sys/asyncLib.h>
int s1dbprintf(unsigned long uart_clock, unsigned char *base_reg, int
reg_delta, event_t event, const char *format,...);
```

## Library

asyncLib.a

## Description

**s1dbprintf()** is a version of **printf()** that uses polled writes (no interrupts), and may be used before I/O has been established. **s1dbprintf()** may be called before the async device driver is installed. *uart\_clock* is the clock frequency of the serial port. *base\_reg* specifies the address of the base UART register. *reg\_delta* specifies the space between UART registers. *event* specifies the external interrupt level. For the 602 EVB, *uart\_clock* must be 3686400, *base\_reg* must be 0x0E900047, *reg\_delta* should be 8

## Errors

None.

## Example

- Print “Hello World” before I/O has been initialized:

```
#include <sys/asyncLib.h>
#define S1DB_PARMS 3686400, (unsigned char *) 0xe900047, 8,
EXT_IRQ_COM1
s1dbprintf(S1DB_PARMS, "Hello World\n\r");
```

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- PowerPC 602 RISC Microprocessor User’s Manual*

## Synopsis

```
#include <sys/asyncLib.h>
int s2dbprintf(unsigned long uart_clock, unsigned char *base_reg, int
reg_delta, event_t event, const char *format,...);
```

## Library

asyncLib.a

## Description

**s2dbprintf()** is a version of **printf()** that uses polled writes (no interrupts), and may be used before I/O has been established. **s2dbprintf()** may be called before the async device driver is installed. *uart\_clock* is the clock frequency of the serial port. *base\_reg* specifies the address of the base UART register. *reg\_delta* specifies the space between UART registers. *event* specifies the external interrupt level. For the 602 EVB, *uart\_clock* must be 3686400, *base\_reg* must be 0x0E900007, *reg\_delta* should be 8

## Errors

None.

## Example

- Print “Hello World” before I/O has been initialized:

```
#include <sys/asyncLib.h>
#define S2DB_PARMS 3686400, (unsigned char *) 0xe900007, 8,
EXT_IRQ_COM2
s2dbprintf(S2DB_PARMS, "Hello World\n\r");
```

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- *PowerPC 602 RISC Microprocessor User’s Manual*

# **timertick\_install()**

---

## **Synopsis**

```
#include <tickLib.h>
int timertick_install(unsigned long ns_adj);
```

## **Library**

tickLib.a

## **Description**

**timertick\_install()** installs and starts the timer tick handler to maintain time-of-day in the OS Open real-time executive.

*ns\_adj* is the number of nano seconds between decrements of the decrementer register (DEC). On the 602 EVB, this value is four times the number of nanoseconds in a bus cycle clock.

## **Errors**

[ENOMEM]	Insufficient memory to install the timer tick handler.
----------	--

## **Example**

- Do a **timertick\_install()** for a 602 processor with a 33Mhz system clock.

```
unsigned long int ns_adj;
/* At 33Mhz there are 30 cycles per nanosecond */
ns_adj=30*4; /* =120 */
timertick_install(ns_adj);
```

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- **timertick\_remove()**, p. 10-101

## **Synopsis**

```
#include <tickLib.h>
int timertick_remove( void );
```

## **Library**

tickLib.a

## **Description**

**timertick\_remove()** removes the timer tick handler installed by **timertick\_install()**.

## **Errors**

[EINVAL]	Internal error involving tick handler level.
----------	--

## **Attributes**

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## **References**

- **timertick\_install()**, p. 10-100

# vs1dbprintf()

---

## Synopsis

```
#include <sys/asyncLib.h>
int vs1dbprintf(unsigned long uart_clock, unsigned char *base_reg,
int reg_delta, event_t event, const char *format,va_list arg_list);
```

## Library

asyncLib.a

## Description

**vs1dbprintf()** is a version of **printf()** that uses polled writes (no interrupts), and may be used before I/O has been established and accepts a va\_list as a parameter instead of a variable number of parameters. **vs1dbprintf()** may be called before the async device driver is installed. *uart\_clock* is the clock frequency of the serial port. *base\_reg* specifies the address of the base UART register. *reg\_delta* specifies the space between UART registers. *event* specifies the external interrupt level. For the 602 EVB, *uart\_clock* must be 3686400, *base\_reg* must be 0x0E900047, *reg\_delta* should be 8

## Errors

None.

## Example

- Print “Hello World” before I/O has been initialized:

```
#include <sys/asyncLib.h>
#define S1DB_PARMS 3686600, (unsigned Char *) 0x900047,8,
EXT_IRQ_COM1
vs1dbprintf(SCLK,"Hello World\n\r");
```

## Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

## References

- PowerPC 602 RISC Microprocessor User’s Manual*



# Program Trace Calls

This appendix describes the remote debugging interface provided by the ROM monitor. These calls may be used by remote debuggers other than the **RISCWatch** debugger provided with the 602 EVB kit.

## A.1 Overview

The following section describes the message (ptrace) protocol that has been implemented in the ROM monitor to support debug. If you want to interface your own debugger to the ROM monitor or modify the ROM monitor to interface with your debugger, you will need to understand the existing message protocol associated with the various debugging functions.

The ptrace interface to the ROM monitor can best be understood by reviewing the information below along with the debug-specific ROM monitor source code (dbLib/ptrace.c).

## A.2 MSGDATA Structure

In the interface descriptions shown below, several references are made to a “process id.” The concept of process ids does not apply to the ROM monitor, so any nonzero value can be used. The ROM monitor uses the value “42”.

Data structure "MSGDATA" is defined in dbg.h. New register definitions and new error messages are also defined in dbg.h file.

### dbg.h File

```
/* @(#)dbg.h 4.3 5/9/95 09:12:14 */
/*
 *-----+
 |   COPYRIGHT    I B M    CORPORATION 1994
 |   LICENSED MATERIAL - PROGRAM PROPERTY OF I B M
 |   REFER TO COPYRIGHT INSTRUCTIONS: FORM G120-2083
 |   US Government Users Restricted Rights - Use, duplication or
 |   disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 *-----*/
#ifndef !defined(DBG_H)
#define DBG_H
#define BREAKPT 0x7D821008
```

```

#ifndef MIN
#define MIN(X,Y) ((X) < (Y) ? (X) : (Y))
#endif

/*ptrace definitions based on AIX ptrace */
#define RD_TRACE_ME      0      /* used ONLY by target task to be traced*/
#define RD_READ_I         1      /* read target instruction addr space */
#define RD_READ_D         2      /* read target data address space */
#define RD_READ_U         3      /* read offset from the user structure */
#define RD_WRITE_I         4      /* write target instruction addr space */
#define RD_WRITE_D         5      /* write target data address space */
#define RD_WRITE_U         6      /* write offset to the user structure */
#define RD_CONTINUE        7      /* continue execution */
#define RD_KILL            8      /* terminate execution */
#define RD_STEP             9      /***execute one or more instructions*** !*/
#define RD_READ_GPR        11     /* read general purpose register */
#define RD_READ_FPR        12     /* read floating point register */
#define RD_WRITE_GPR        14     /* write general purpose register */
#define RD_WRITE_FPR        15     /* write floating point register */
#define RD_READ_BLOCK       17     /* read block of data */
#define RD_WRITE_BLOCK      19     /* write block of data */
#define RD_ATTACH           30     /* attach to a process */
#define RD_DETACH           31     /* detach a proc to let it keep running */
#define RD_REGSET           32     /* return entire register set to caller */
#define RD_REATT             33     /* reattach debugger to proc */
#define RD_LDINFO            34     /* return loaded program info */
#define RD_MULTI             35     /* set/clear multi-processing */
#define RD_READ_I_MULT       70     /* Read multiple inst words */
#define RD_READ_GPR_MULT    71     /* Read multiple registers */
#define RD_SINGLE_STEP      100    /***source line single step***** !*/
#define RD_LOAD              101    /* load a task */
#define RD_LOGIN             103    /*ptrace for login */
#define RD_LOGON             103    /*ptrace for logon */
#define RD_LOGOFF            104    /*ptrace for logoff */
#define RD_FILL               105    /*ptrace for fill memory */
#define RD_PASS              106    /*ptrace for pass */
#define RD_SEARCH            107    /*ptrace for search memory */
#define RD_WAIT              108    /*ptrace for wait status information */
/* Added to support ADEPT */
#define RD_READ_DCR          110    /*ptrace for reading DCR's */
#define RD_WRITE_SPR          111    /*ptrace for writing SPR's */
#define RD_WRITE_DCR          112    /*ptrace for writing DCR's */
#define RD_STOP_APPL          113    /*ptrace for stopping the application */
#define RD_STATUS             114    /*ptrace for getting run status */
#define RD_READ_SPR           115    /*ptrace for reading SPR's */
/* Added to support 403GC */
#define RD_READ_TLB            116    /*ptrace for readingTLB(403GC ) */
#define RD_WRITE_TLB           117    /*ptrace for writing TLB(403GC ) */
/* Added to support 602 */
#define RD_READ_SR             118    /*ptrace for reading SR's */
#define RD_WRITE_SR            119    /*ptrace for writing SR's */

```

```

#define MAX_PTRACE      119      /*last ptrace number          */
#define RL_LOAD_REQ     180      /* Remote Loader - Load Request   */
#define RL_LDINFO       181      /* Remote Loader - Load Information  */
/*TCP/IP services for all sorts of remote debug
#define OSOPEN_SERVNAME "osopen-dbg" /* OS/Open debug service
#define OSOPEN_MON_SERVNAME "osopen-mon" /* OS/Open debug monitor svc
/*new register definition
#define DAR    137      /* Data Address Register ($dar)      */
#define DSISR  138      /* Data St Int Status Reg ($dsisr)    */
#define SRR0   139      /* Save and Restore Register 0 ($srr0)    */
#define SRR1   140      /* Save and Restore Register 0 ($srr1)    */
#define SR0    141      /* Segment Register ($sr0)      */
#define SR1    142      /* Segment Register ($sr1)      */
#define SR2    143      /* Segment Register ($sr2)      */
#define SR3    144      /* Segment Register ($sr3)      */
#define SR4    145      /* Segment Register ($sr4)      */
#define SR5    146      /* Segment Register ($sr5)      */
#define SR6    147      /* Segment Register ($sr6)      */
#define SR7    148      /* Segment Register ($sr7)      */
#define SR8    149      /* Segment Register ($sr8)      */
#define SR9    150      /* Segment Register ($sr9)      */
#define SR10   151      /* Segment Register ($sr10)     */
#define SR11   152      /* Segment Register ($sr11)     */
#define SR12   153      /* Segment Register ($sr12)     */
#define SR13   154      /* Segment Register ($sr13)     */
#define SR14   155      /* Segment Register ($sr14)     */
#define SR15   156      /* Segment Register ($sr15)     */
#define DEC    157      /* Decrementer ($dec)      */
#define RTCU   158      /* Real Time Clock Upper ($rtcu)    */
#define RTCL   159      /* Real Time Clock Lower ($rtcl)    */
#define SDR0   160      /* Storage Description Reg ($sdr0)    */
#define SDR1   161      /* Storage Description Reg ($sdr1)    */
#define EISO   162      /* External Int Summary Reg1($eis1)    */
#define EIS1   163      /* External Int Summary Reg2($eis2)    */
#define EIMO   164      /* External Int Mask Reg1($eim1)    */
#define EIM1   165      /* External Int Mask Reg2($eim2)    */
#define SRR2   166      /* Save and Restore Register 2 ($srr2)    */
#define SRR3   167      /* Save and Restore Register 3 ($srr3)    */
/*other definitions needed for remote debug
#define RD_MAXDATA    1800      /* Total no of DWORDS in a MSGDATA   */
#define RD_MINLENGTH  6      /* Min no of dwords in msg        */
#define RD_MINBYTES  (RD_MINLENGTH*sizeof(unsigned long))
#define RD_MAXBUFFER (RD_MAXDATA - RD_MINLENGTH)
#define RD_MAXPACKET 1000000    /* Max bytes in TCP/IP packet      */
#define RD_REGBYTES  (32+8)*4    /* No of bytes for all registers   */
#define NO_KILL        1      /* /*do not kill any users processes */
#define KILL_PROC      0      /* /*kill user process upon logoff   */
#define MAX_ERROR      1014     /* /*last error for rptrace        */
#define MIN_ERROR      1000     /* /*first error for rptrace       */
#define MIN_PACKET_SIZE 24

```

```

#define DBG_SPORT    20044
#define DBG_DPORT    20050
/*new error codes
#define RD_NOLOAD_ERR    1000    /*no loader info available      */
#define RD_COM_ERR       1001    /*communication error occurred   */
#define RD_SIZE_ERR      1002    /*not enough room to pass all info   */
#define RD_NOTSUPP       1003    /*call not supported           */
#define RD_REG_ERR       1004    /*invalid register number requested */
#define RD_NOTAVAIL      1005    /*call not implemented at this time */
#define RD_NOFILE_ERR    1006    /*file could not be loaded, no file   */
#define RD_NOSCAN_ERR    1008    /*could not locate scan string file */
#define RD_NOPERM        1010    /*no permission to log on          */
#define RD_INVALID_SEQ   1011    /*invalid rptrace sequence        */
#define RD_BUSY_ERR      1012    /*some users is already logged on   */
#define RD_PTRACE_ERR    1014    /*internal ptrace error          */
#define RD_OK             0       /*rptrace completed ok           */
#define ARCH_403          0x34000000    /* 403 architecture */
#define ARCH_601          0x36000000    /* 601 architecture */
#define ARCH_602          0x36303200    /* 602 architecture */
#define ARCH_603          0x36303300    /* 603 architecture */
#define ARCH_604          0x36303400    /* 604 architecture */
typedef struct msgdata
{
    unsigned long data_len;           /* message data structure      */
    unsigned long retcode;           /* optional data length        */
    unsigned long request;           /* return code                 */
    unsigned long address;           /* request type               */
    unsigned long data;              /* function parameter          */
    struct {
        unsigned f1:1;
        unsigned f2:1;
        unsigned f3:1;
        unsigned padd:21;
        unsigned f25:8;
    } flags;
#define printmsg flags.f1
#define breakpt flags.f2
#define dbg_seqno flags.f25
union {
    unsigned long trace_buffer[RD_MAXBUFFER];
    unsigned long processid;
} parameter;
#define buffer parameter.trace_buffer /* buffer for data, in any      */
#define rpid parameter.processid     /* process id                  */
} MSGDATA;
#endif

```

### A.3 Ptrace Definitions

The following section presents the application programming interface (API) for rptrace messages. One field that is not shown here, because it is common to every call, is the *msg.printmsg* flag. This may be set in an rptrace response where *msg.recode* does not equal

RD\_OK. When the *msg.printmsg* flag is set it indicates that a text string is contained in *msg.buffer* and that this message should be displayed to the user. Typically this is an error message that provides more detail as to why the rptrace call failed to return RD\_OK.

Another field that is not shown is the *dbg\_seqno* field. The field provides a mechanism for recovering from lost requests and responses. If a request has the *dbg\_seqno* field as not zero, it is compared with the value from the previous request. If it matches, the action is not performed and instead, the previous response is sent. This allows the debugger to time-out and re-try requests without danger of performing the same function twice.

### A.3.1 RD\_ATTACH (30)

Attaches debugger to running process in target environment.

#### A.3.1.1 Request data

Table A-1. RD\_ATTACH Request Table

Parameters	Description
msg.request= RD_ATTACH	Requested API function.
msg.rpid= process_id	Numeric process ID on the target system.(Any non zero value)
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

#### A.3.1.2 Response data

Table A-2. RD\_ATTACH Response Table

Parameters	Description
msg.retcode= ESRCH (3)	The <i>msg.pid</i> parameter identifies a process that does not exist.
msg.retcode= EIO (5)	One of the parameters is incorrect.
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_NOTSUPP (1003)	Call not supported for this interface.
msg.retcode= RD_OK (0)	Successful completion.
msg.data_len=0	No additional data

### A.3.2 RD\_CONTINUE (7)

This request causes the process to resume execution. If the *dbg\_seqno* field of the request is zero, the response is not returned until the process stops due to a breakpoint or error. Otherwise, an immediate response is sent from the **RD\_CONTINUE** request and the debugger should send the **RD\_STATUS** request to see if the process has stopped.

#### A.3.2.1 Request data

**Table A-3. RD\_CONTINUE Request Table**

Parameters	Description
msg.request= RD_CONTINUE	Requested API function.
msg.address= address	This field is ignored by ROM monitor.
msg.data= signal	0
msg.rpid= process_id	Numeric process ID on the target system.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

#### A.3.2.2 Response data

**Table A-4. RD\_CONTINUE Response Table**

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.data= 0	

### A.3.3 RD\_DETACH (31)

Detaches debugger from running process in target environment. Debugged process is restarted and execution continues without debugger control.

#### A.3.3.1 Request data

Table A-5. RD\_DETACH Request Table

Parameters	Description
msg.request= RD_DETACH	Requested API function.
msg.rpid= process_id	Numeric process ID on the target system.
msg.data= 0	Ignored by ROM monitor.
msg.address=1	Ignored by ROM monitor.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

#### A.3.3.2 Response data

Table A-6. RD\_DETACH Response Table

Parameters	Description
msg.retcode= ESRCH (3)	The <i>msg.pid</i> parameter identifies a process that does not exist, or a process that is currently not being debugged.
msg.retcode= RD_COM_ERR (1001)	Communications error occurred.
msg.retcode= RD_NOTSUPP (1003)	Call not supported for this interface.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= EIO (5)	One of the parameters is incorrect.
msg.data_len= 0	No additional data is being sent.

### A.3.4 RD\_FILL (105)

Fills memory with zeroes at the location specified by *address* for the number of bytes specified by *data*.

#### A.3.4.1 Request data

**Table A-7. RD\_FILL Request Table**

Parameters	Description
msg.request= RD_FILL	Requested API function.
msg.rpid= process_id	Numeric process ID on the target system.
msg.address= address	Address of memory to fill with zeroes
msg.data= count	Number of bytes to fill with zeroes
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

#### A.3.4.2 Response data

**Table A-8. RD\_FILL Response Table**

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communications error occurred.
msg.retcode= RD_NOTSUPP (1003)	Call not supported for this interface.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= EIO (5)	One of the parameters is incorrect.
msg.data_len= 0	No additional data is being sent.

### A.3.5 RD\_KILL (8)

This request causes the process to terminate the same way it would with an exit routine. The ROM monitor does not implement this function but simply returns an **RD\_OK** response for compatibility with older debuggers.

#### A.3.5.1 Request data

**Table A-9. RD\_KILL Request Table**

Parameters	Description
msg.request= RD_KILL	Requested API function.
msg.rpid= process_id	Process ID of the process to be killed.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

#### A.3.5.2 Response data

**Table A-10. RD\_KILL Response Table**

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= ESRCH (3)	The <i>msg.pid</i> parameter identifies a process that does not exist.
msg.data_len= 0	Length of additional data being sent.

## A.3.6 RD\_LDINFO (34)

Request loader information from target environment. This information is provided to the ROM monitor in the boot header or by the **RL\_LDINFO** request. Refer to **ROM Monitor Load Format** section for more information.

### A.3.6.1 Request data

Table A-11. RD\_LDINFO Request Table

Parameters	Description
msg.request= RD_LDINFO	Requested API function.
msg.rpid= process_id	Process ID from which the loader information is requested.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

### A.3.6.2 Response data

Table A-12. RD\_LDINFO Response Table

Parameters	Description
msg.retcode= RD_NOLOAD_ERR (1000)	No loader information is available.
msg.retcode= ESRCH (3)	The <i>msg.pid</i> parameter identifies a process that does not exist.
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_SIZE_ERR (1002)	Not enough room in the buffer to fit all load information.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= EIO (5)	One of the parameters is incorrect.
msg.buffer[0]= Idinfo_next	Offset to next loader information segment. See note below.
msg.buffer[1]= fd	File descriptor for loaded object. In remote debug 0xFFFF FFFF should be returned (this is a space filler).

**Table A-12. RD\_LDINFO Response Table**

Parameters	Description
msg.buffer[2]= textorig	Starting text address.
msg.buffer[3]= textsizE	Size of text.
msg.buffer[4]= dataorig	Starting data address.
msg.buffer[5]= datasizE	Size of data.
msg.buffer[6]= (char *)pathname	Fully qualified filename of the object file.
msg.buffer[X]= (char *)membername	Member name (used for shared library objects). <b>X</b> does not represent position on word boundary. A NULL has to be returned for the membername even if the debugged file has no membername.
msg.buffer[linfo_next]= linfo_next	Next loader block (notice "linfo_next").
msg.data_len= "variable"	Set to length of data sent in msg.buffer. Data length will vary depending on the amount of information passed. Remember to count all the NULL characters.
Note: <i>linfo_next=0</i> indicates that no further loader blocks are present, otherwise <i>linfo_next</i> contains the offset of the next loader block in the buffer. This is actually the length of the current block. For example, if the buffer contains three blocks of lengths 38, 40 and 41 bytes, the <i>linfo_next</i> fields would be 38, 40 and 0, respectively. Note also that the blocks do not have to be contiguous - it is possible that the end of one block may not directly abut the following block. This may occur if additional information or word-aligning padding is placed after the end of the member-name string. Path-name and member-name are strings terminated with a NULL character.	

### A.3.7 RD\_LOAD (101)

Loads executable program. Full path name of the file to be loaded is passed in this message. The ROM monitor will respond by sending an **RL\_LOAD\_REQ** to the remote loader daemon port.

#### A.3.7.1 Request data

**Table A-13. RD\_LOAD Request Table**

Parameters	Description
msg.request= RD_LOAD	Requested API function.
msg.buffer= filename	Name of file to load. A NULL character terminates filename. Filename contains fully qualified path to that file.
msg.data_len= strlen(filename)+1	String length of filename plus NULL character.

#### A.3.7.2 Response data

**Table A-14. RD\_LOAD Response Table**

Parameters	Description
msg.retcod= RD_COM_ERR (1001)	Communication error occurred.
msg.retcod= RD_OK (0)	Successful completion.
msg.retcod= RD_NOFILE_ERR (1006)	Could not locate/load the file.
msg.rpid= process_id	Process_id of the newly loaded file. This number (integer) can not be equal to -1 (0xFFFF FFFF) or 0.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

## A.3.8 RD\_LOGIN (103)

Initializes users LOGIN. This request must be the first rptrace request issued by the debugger or results will be unpredictable.

### A.3.8.1 Request data

Table A-15. RD\_LOGIN Request Table

Parameters	Description
msg.request= RD_LOGIN	Requested API function.
msg.buffer[0]= host_name	This field is ignored by ROM monitor.
msg.buffer[strlen(host_name)+1]= user_name	This field is ignored by ROM monitor.
msg.data_len= strlen(host_name)+strlen(user_name)+2	Length of additional data being sent.

### A.3.8.2 Response data

Table A-16. RD\_LOGIN Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.data_len= 0	Length of additional data being sent.

### A.3.9 RD\_LOGOFF (104)

Performs user LOGOFF function. This is used when the debugger performs normal termination using quit or detach.

#### A.3.9.1 Request data

Table A-17. RD\_LOGOFF Request Table

Parameters	Description
msg.request= RD_LOGOFF	Requested API function.
msg.data= NO_KILL	This field is ignored by ROM monitor.
msg.data_len= 0	Length of additional data being sent.

#### A.3.9.2 Response data

Table A-18. RD\_LOGOFF Response Table

Parameters	Description
msg.retcod= RD_COM_ERR (1001)	Communication error occurred.
msg.retcod= RD_OK (0)	Successful completion.
msg.retcod= RD_INVALID_SEQ (1011)	Not logged on.
msg.data_len= 0	Length of additional data being sent.

### A.3.10 RD\_READ\_D (2)

This request returns the integer in the debugged process address space at the location pointed to by the *address* parameter. If the value of *address* is not in a valid address space, unpredictable results will occur.

#### A.3.10.1 Request data

**Table A-19. RD\_READ\_D Request Table**

Parameters	Description
msg.request= RD_READ_D	Requested API function.
msg.address= address	Address of memory to read data from.
msg.rpid= process_id	Numeric process ID on the target system.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

#### A.3.10.2 Response data

**Table A-20. RD\_READ\_D Response Table**

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= EIO (5)	Debugged process can not access given address.
msg.retcode= ESRCH (3)	The <i>msg.pid</i> parameter identifies a process that does not exist.
msg.data= data	Data read at location pointed to by address. -1 if error.
msg.data_len= 0	Length of additional data being sent.

## A.3.11 RD\_READ\_FPR (12)

This request returns the content of one of the floating-point registers.

### A.3.11.1 Request data

**Table A-21. RD\_READ\_FPR Request Table**

Parameters	Description
msg.request= RD_READ_FPR	Requested API function.
msg.rpid= process_id	Numeric process ID on the target system.
msg.address= register	Name of the register to be read.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

### A.3.11.2 Response data

**Table A-22. RD\_READ\_FPR Response Table**

Parameters	Description
msg.retcod= RD_COM_ERR (1001)	Communication error occurred.
msg.retcod= RD_OK (0)	Successful completion.
msg.retcod= EIO (5)	Register is not defined.
msg.retcod= RD_REG_ERR (1004)	Unable to access given register.
msg.data= value	Value read from register. 0xFFFFFFFF if error occurred.
msg.retcod= ESRCH (3)	The <i>msg.pid</i> parameter identifies a process that does not exist.
msg.data_len= 0	Length of additional data being sent.

### A.3.12 RD\_READ\_GPR (11)

This request returns the content of one of the general-purpose or special-purpose registers of the debugged process. Valid registers are defined in "dbg.h" and "sys/reg.h". Not all defined registers are supported for all environments.

#### A.3.12.1 Request data

Table A-23. RD\_READ\_GPR Request Table

Parameters	Description
msg.request= RD_READ_GPR	Requested API function.
msg.rpid= process_id	Numeric process ID on the target system.
msg.address= register	Name of the register to be read.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

#### A.3.12.2 Response data

Table A-24. RD\_READ\_GPR Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= EIO (5)	Register is not defined.
msg.retcode= RD_REG_ERR (1004)	Unable to access given register.
msg.data= value	Value read from register. 0xFFFFFFFF if error occurred.
msg.retcode= ESRCH (3)	The <i>msg.pid</i> parameter identifies a process that does not exist.
msg.data_len= 0	Length of additional data being sent.

### A.3.13 RD\_READ\_GPR\_MULT(71)

This request returns the contents of general-purpose registers 0 to 18, inclusive, of the debugged process.

#### A.3.13.1 Request data

Table A-25. RD\_READ\_GPR\_MULT Request Table

Parameters	Description
msg.request= RD_READ_GPR_MULT	Requested API function.
msg.rpid= process_id	Numeric process ID on the target system.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

#### A.3.13.2 Response data

Table A-26. RD\_READ\_GPR\_MULT Response Table

Parameters	Description
msg.retcod= RD_COM_ERR (1001)	Communication error occurred.
msg.retcod= RD_OK (0)	Successful completion.
msg.retcod= RD_NOTSUPP (1003)	Call not supported by this interface.
msg.retcod= RD_REG_ERR (1004)	Unable to access given register.
msg.retcod= ESRCH (3)	The <i>msg.pid</i> parameter identifies a process that does not exist.
msg.data_len= 76 (0x4C)	Length of additional data being sent.
msg.buffer[0-18]	Values read from GPR0 to GPR18. Undefined if error.

### A.3.14 RD\_READ\_I (1)

This request returns the integer in the debugged process address space at the location pointed to by the *address* parameter. If the value of *address* is not in a valid address space, unpredictable results will occur.

#### A.3.14.1 Request data

**Table A-27. RD\_READ\_I Request Table**

Parameters	Description
msg.request= RD_READ_I	Requested API function.
msg.address= address	Address of memory to read data from.
msg.rpid= process_id	Numeric process ID on the target system.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

#### A.3.14.2 Response data

**Table A-28. RD\_READ\_I Response Table**

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= EIO (5)	Debugged process can not access given address.
msg.retcode= ESRCH (3)	The <i>msg.pid</i> parameter identifies a process that does not exist.
msg.data= data	Data read at location pointed to by address. -1 if error (retcode should also be set to EIO).
msg.data_len= 0	Length of additional data being sent.

## A.3.15 RD\_READ\_I\_MULT (71)

This request returns the 32 integers in the debugged process address space at the location pointed to by the *address* parameter. If the value of *address* is not in a valid address space, unpredictable results will occur.

### A.3.15.1 Request data

Table A-29. RD\_READ\_I\_MULT Request Table

Parameters	Description
msg.request= RD_READ_I_MULT	Requested API function.
msg.address= address	Address of memory to read data from.
msg.rpid= process_id	Numeric process ID on the target system.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

### A.3.15.2 Response data

Table A-30. RD\_READ\_I\_MULT Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= EIO (5)	Debugged process can not access given address.
msg.retcode= ESRCH (3)	The <i>msg.pid</i> parameter identifies a process that does not exist.
msg.retcode= RD_NOTSUPP (1003)	Call not supported by this interface.
msg.buffer[0-0x1F]	Contents of addresses from location pointed to by address to address + 0x1F.
msg.data_len= 128 (0x80)	Length of additional data being sent.

## A.3.16 RD\_READ\_SPR (115)

This request reads data directly from one of the SPRs (not the process's copy). All SPR registers are accessible through this message request. The sender is responsible for supplying valid SPR values, no error checking is performed on this field.

### A.3.16.1 Request data

Table A-31. RD\_READ\_SPR Request Table

Parameters	Description
msg.request= RD_READ_SPR	Requested API function.
msg.address= SPR number	SPR number to read.
msg.data_len= 0	Length of additional data being sent.

### A.3.16.2 Response data

Table A-32. RD\_READ\_SPR Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.data= value	Value read from register.
msg.data_len= 0	Length of additional data being sent.

## A.3.17 RD\_READ\_SR (118)

This request returns the content of one of the segment registers.

### A.3.17.1 Request data

Table A-33. RD\_READ\_SR Request Table

Parameters	Description
msg.request= RD_READ_SR	Requested API function.
msg.rpid= process_id	Numeric process ID on the target system.
msg.address= register	Name of the register to be read.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

### A.3.17.2 Response data

Table A-34. RD\_READ\_SR Response Table

Parameters	Description
msg.retcod= RD_COM_ERR (1001)	Communication error occurred.
msg.retcod= RD_OK (0)	Successful completion.
msg.retcod= EIO (5)	Register is not defined.
msg.retcod= RD_REG_ERR (1004)	Unable to access given register.
msg.data= value	Value read from register. 0xFFFFFFFF if error occurred.
msg.retcod= ESRCH (3)	The <i>msg.pid</i> parameter identifies a process that does not exist.
msg.data_len= 0	Length of additional data being sent.

## A.3.18 RD\_STATUS (114)

This request is used to get program execution status and to determine if a previous RD\_CONTINUE request was received.

### A.3.18.1 Request data

Table A-35. RD\_STATUS Request Table

Parameters	Description
msg.request= RD_STATUS	Requested API function.
msg.rpid= process_id	Numeric process ID on the target system
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

### A.3.18.2 Response data

Table A-36. RD\_STATUS Response Table

Parameters	Description
msg.address= execution status	Status is 1 if program is running and 0 if stopped. In the case of an error, this field will be -1 (0xFFFFFFFF).
msg.data= sequence number	Sequence number of the last RD_CONTINUE request that was received.
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= RD_ESRCH (3)	The msg.pid field identifies a process that does not exist.

## A.3.19 RD\_STOP\_APPL (113)

This request is used to interrupt program execution.

### A.3.19.1 Request data

**Table A-37. RD\_STOP\_APPL Request Table**

Parameters	Description
msg.request= RD_STOP_APPL	Requested API function.
msg.rpid= process_id	Numeric process ID on the target system
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

### A.3.19.2 Response data

**Table A-38. RD\_STOP\_APPL Response Table**

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= RD_ESRCH (3)	The msg.pid field identifies a process that does not exist.

### A.3.20 RD\_WAIT (108)

This call allows the debugger to determine the current status of the debugged process after it is stopped. The first (least significant) byte of the process status indicates the reason for stoppage: this is always 0x7f. The second byte contains the signal number that caused the stop. Valid signals are:

- AIX\_SIGILL (4) - illegal instruction
- AIX\_SIGTRAP (5) - hit a trap instruction (breakpoint)
- AIX\_SIGFPE (8) - floating point error
- AIX\_SIGSEGV (11) - storage violation

For example after hitting a breakpoint, the status of 0x57f is returned to the debugger. After the program terminates, the first byte contains 0x00 and the rest of the status holds the program exit code. After RD\_KILL call wait status of 0x57f should be returned.

#### A.3.20.1 Request data

Table A-39. RD\_WAIT Request Table

Parameters	Description
msg.request= RD_WAIT	Requested API function.
msg.data_len= 0	Length of data in msg.buffer.

#### A.3.20.2 Response data

Table A-40. RD\_WAIT Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.data= status	Process status.
msg.address= pid	Process id.
msg.data_len= strlen(message_string)	The ROM monitor always returns 0 in this field.
msg.buffer= message_string	Formatted message string text (NULL terminated).

### A.3.21 RD\_WRITE\_BLOCK (19)

This request writes a block of data into the address space of the debugged process at the address pointed to by the *msg.address* field. The number of bytes to write is contained in the *msg.data* field and the data is in the *msg.buffer* field. Unpredictable results occur if the *msg.address* parameter points to a location that can not be accessed by the debugged process.

#### A.3.21.1 Request data

**Table A-41. RD\_WRITE\_BLOCK Request Table**

Parameters	Description
msg.request= RD_WRITE_BLOCK	Requested API function.
msg.address= address	Address of memory to write data to.
msg.data= count	Number of bytes of buffer area to be written
msg.buffer	Data to be written.
msg.data_len= count	Length of additional data being sent.

#### A.3.21.2 Response data

**Table A-42. RD\_WRITE\_BLOCK Response Table**

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= EIO (5)	Debugged process can not access given address.
msg.data_len= 0	Length of additional data being sent.

## A.3.22 RD\_WRITE\_D (5)

This request writes the value of the *msg.data* parameter into the address space of the debugged process at the address pointed to by the *msg.address* parameter. Unpredictable results occur if the *msg.address* parameter points to a location that can not be accessed by the debugged process.

### A.3.22.1 Request data

Table A-43. RD\_WRITE\_D Request Table

Parameters	Description
msg.request= RD_WRITE_D	Requested API function.
msg.address= address	Address of memory to write data to.
msg.data= data	Data to write to memory.
msg.rpid= process_id	Numeric process ID on the target system.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

### A.3.22.2 Response data

Table A-44. RD\_WRITE\_D Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= EIO (5)	Debugged process can not access given address.
msg.retcode= ESRCH (3)	The <i>msg.pid</i> parameter identifies a process that does not exist.
msg.data= data	Data written at location pointed to by address. -1 if error (retcode should also be set to EIO or ESRCH).
msg.data_len= 0	Length of additional data being sent.

### A.3.23 RD\_WRITE\_FPR (15)

This request writes data to one of the floating-point registers.

#### A.3.23.1 Request data

**Table A-45. RD\_WRITE\_FPR Request Table**

Parameters	Description
msg.request= RD_WRITE_FPR	Requested API function.
msg.rpid= process_id	Numeric process ID on the target system.
msg.address= register	Name of the register to be written.
msg.data= value	Value to be written to the register.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

#### A.3.23.2 Response data

**Table A-46. RD\_WRITE\_FPR Response Table**

Parameters	Description
msg.retcod= RD_COM_ERR (1001)	Communication error occurred.
msg.retcod= RD_OK (0)	Successful completion.
msg.retcod= EIO (5)	Register is not defined.
msg.retcod= RD_REG_ERR (1004)	Unable to access given register.
msg.data= value	Value written to register. 0xFFFFFFFF if error occurred.
msg.retcod= ESRCH (3)	The <i>msg.pid</i> parameter identifies a process that does not exist.
msg.data_len= 0	Length of additional data being sent.

## A.3.24 RD\_WRITE\_GPR (14)

This request writes data to one of the general-purpose or special-purpose registers of the debugged process. Valid registers are defined in "dbg.h" and "sys/reg.h". Not all defined registers are supported for all environments.

### A.3.24.1 Request data

**Table A-47. RD\_WRITE\_GPR Request Table**

Parameters	Description
msg.request= RD_WRITE_GPR	Requested API function.
msg.rpid= process_id	Numeric process ID on the target system.
msg.address= register	Name of the register to be written.
msg.data= value	Value to be written to the register.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

### A.3.24.2 Response data

**Table A-48. RD\_WRITE\_GPR Response Table**

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= EIO (5)	Register is not defined.
msg.retcode= RD_REG_ERR (1004)	Unable to access given register.
msg.data= value	Value written to register. 0xFFFFFFFF if error occurred.
msg.retcode= ESRCH (3)	The <i>msg.pid</i> parameter identifies a process that does not exist.
msg.data_len= 0	Length of additional data being sent.

### A.3.25 RD\_WRITE\_I (4)

This request writes the value of the *msg.data* parameter into the address space of the debugged process at the address pointed to by the *msg.address* parameter. This request fails if the *msg.address* parameter points to a location that can not be accessed by debugged process. This call sets break points in the debugged process by writing TRAP (0x7D821008) instructions.

#### A.3.25.1 Request data

**Table A-49. RD\_WRITE\_I Request Table**

Parameters	Description
msg.request= RD_WRITE_I	Requested API function.
msg.rpid= process_id	Numeric process ID on the target system.
msg.address= address	Address of memory to write data to.
msg.data= data	Data to write to memory.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

#### A.3.25.2 Response data

**Table A-50. RD\_WRITE\_I Response Table**

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= EIO (5)	Debugged process can not access given address.
msg.retcode= ESRCH (3)	The <i>msg.pid</i> parameter identifies a process that does not exist.
msg.data= data	Data written at location pointed to by address. -1 if error (retcode should also be set to EIO or ESRCH).
msg.data_len= 0	Length of additional data being sent.

### A.3.26 RD\_WRITE\_SPR (112)

This request writes data directly to one of the SPRs (not the process's copy). All SPR registers are accessible through this request. The requester is responsible for supplying valid SPR values. No error checking is performed on this field.

#### A.3.26.1 Request data

**Table A-51. RD\_WRITE\_SPR Request Table**

Parameters	Description
msg.request= RD_WRITE_SPR	Requested API function.
msg.address= SPR number	SPR number to be written
msg.data= value	Data to write to register.
msg.data_len= 0	Length of additional data being sent.

#### A.3.26.2 Response data

**Table A-52. RD\_WRITE\_SPR Response Table**

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.data_len= 0	Length of additional data being sent.

## A.3.27 RD\_WRITE\_SR (119)

This request writes data to one of the segment registers.

### A.3.27.1 Request data

**Table A-53. RD\_WRITE\_SR Request Table**

Parameters	Description
msg.request= RD_WRITE_SR	Requested API function.
msg.rpid= process_id	Numeric process ID on the target system.
msg.address= register	Name of the register to be written.
msg.data= value	Value to be written to the register.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

### A.3.27.2 Response data

**Table A-54. RD\_WRITE\_SR Response Table**

Parameters	Description
msg.retcod= RD_COM_ERR (1001)	Communication error occurred.
msg.retcod= RD_OK (0)	Successful completion.
msg.retcod= EIO (5)	Register is not defined.
msg.retcod= RD_REG_ERR (1004)	Unable to access given register.
msg.data= value	Value written to register. 0xFFFFFFFF if error occurred.
msg.retcod= ESRCH (3)	The <i>msg.pid</i> parameter identifies a process that does not exist.
msg.data_len= 0	Length of additional data being sent.

## A.3.28 RL\_LDINFO (181)

This request provides load information from the host to the ROM monitor. This request is used when the target is loaded by a process other than the debugger. The information specified on this request will be returned on subsequent **RD\_LDINFO** requests.

### A.3.28.1 Request data

Table A-55. RL\_LDINFO Request Table

Parameters	Description
msg.request= RL_LDINFO	Requested API function.
msg.data_len= sizeof(struct Idinfo) + strlen(pathname)	Length of additional data being sent.
msg.buffer= load information	See description of RD_LDINFO request.

### A.3.28.2 Response data

Table A-56. RL\_LDINFO Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.data_len= 0	Length of additional data being sent.

### A.3.29 RL\_LOAD\_REQ(180)

This request flows from the ROM monitor to the host when a RD\_LOAD request is received. The port of the request is for the remote loader daemon (20050) to accommodate loading by a process independent from the debugger.

#### A.3.29.1 Request data

Table A-57. RL\_LOAD\_REQ Request Table

Parameters	Description
msg.request= RL_LOAD_REQ	Requested API function.
msg.buffer= filename	NULL terminated string containing fully qualified name of file to be loaded.
msg.data_len= strlen(filename)	Length of additional data being sent.

#### A.3.29.2 Response data

Table A-58. RL\_LOAD\_REQ Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= RD_NOFILE_ERR (1006)	Can't open file or file is incorrect format.
msg.retcode= RD_PTRACE_ERR (1014)	Error reading file.
msg.rpid= process_id	Process ID of newly loaded file. This number (integer) can not be equal to -1 (0xFFFF FFFF) or 0.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.



# ROM Monitor Load Format

---

---

This appendix presents the ROM Monitor load format requirements.

## B.1 Overview

The ROM Monitor load format is designed to permit the specification of multiple text and data sections. The format consists of a linked list of sections of specified types prefixed by a small boot header, *boot\_block*, that specifies the initial target of the image and the entry point. The *boot\_block* header is placed at the front of the image by **eimgbld** or **nimgbld**. The ROM Monitor does no relocation. It is assumed that the destination addresses for the individual sections are the same ones specified during the application's linkage. The *info\_block* structure is reserved in the bootstrap program, bootLib.s. **nimgbld** or **eimgbld** patch in the values within the *info\_block* structure for bootLib to use at run time. The bootstrap program processes the sections back to front, that is, from the end of the image to the beginning. This is to avoid destructive overlap during the processing of typical images.

The sections are preceded by header blocks which identify the section types. The headers are linked together in a doubly linked list.

## B.2 Section Types

There are three basic section types. Generally, they can occur in the image in any order, but are usually arranged in ascending address order. The section header block has the following format:

```
/*-----+
| Relocation block structure.
+-----*/
typedef struct rel_block {
    unsigned long      type;
    unsigned long      dest_addr;
    unsigned long      size;
    union {
        struct data_info {
            unsigned long  size_to_fill;
            unsigned long  char_to_fill;
```

```

} data_info_str;
struct text_info {
    unsigned long toc_pointer; /* used for XCOFF; not used for ELF */
    unsigned long entry_pt;
} text_info_str;
unsigned long number_symbols;
} section_info;
struct rel_block *next;
struct rel_block *bptr;
} rel_block_t;

```

The **type** field is one of the following manifest constants:

```

#define TEXT_SECT    0x00000001
#define DATA_SECT    0x00000002
#define SYMB_SECT    0x00000004

```

The **dest\_addr** specifies the target for the block, while **size** is the extent of the block, not counting the header. The bootstrap program uses this information to move the block to the destination specified at link time. **next** and **bptr** are the section header forward and backward pointers, respectively.

### B.2.1 First Section

The first section is a text section. The ROM loader places the entire image at the address specified in the *boot\_block* header. The entry point specified in the *boot\_block* header is assumed to be a branch, followed by the first section header, *info\_block*. This is to allow the bootstrap to easily gain immediate addressability to the first section block.

The format of the first section block is shown below:

```

/*-----+
| First section header
+-----*/
struct info_block {
    long magic_num;                      /* magic number          */
    long text_start;                     /* addr of text section from section header */
    long text_size;                      /* size of text section from section header */
    long data_start;                     /* addr of data section from section header */
    long data_size;                      /* size of data section from section header */
    long elf_hdr_size;                  /* size of ELF headr      */
    long sym_start;                     /* addr of symbol table   */
    long num_syms;                      /* number of symbols      */
    long toc_ptr;                       /* used for XCOFF; not used for ELF */
    struct rel_block * next;            /* pointer to next boot section header */
};

```

- **magic\_num** is used for verification purposes and must be X'004D 5054'.
- **text\_start** is the physical address value from the object text header.
- **text\_size** is the size in bytes from the object text header.
- **data\_start** is the physical address from the object data header.

- **data\_size** is the size in bytes from the object data header.
- **elf\_hdr\_size** is the size of the object header. The debugger requires this information.
- **sym\_start** is the address of the symbol table in storage.
- **num\_syms** is the number of symbol entries.
- **next** points to the next section header.

### B.2.2 Text Section

For a text section, the union **section\_info** contains the structure **text\_info**, specifying the entry point of the text section.

### B.2.3 Data Section

For a data section, the union **section\_info** contain the structure **data\_info**, specifying **size\_to\_fill** and **char\_to\_fill**. These parameters are used to optionally fill a region past the size extent specified in the base rel\_block with a character. It is most often used to zero bss by specifying the size of the bss in **size\_to\_fill** and 0x0 for **char\_to\_fill**.

### B.2.4 Symbol Section

For symbols, the union **section\_info** contains the number of symbols in the section. The data in this section consists of the symbol table from the original object file.

## B.3 Boot Header

The entire image is preceded by the boot header that was added by **nimgbld** or **eimgbld**. The ROM loader uses this information to verify that it is a ROM Monitor load image, determine where to place the image, and whether to invoke the ROM Monitor debugger before transferring control to the entry point. The boot header is stripped off by the ROM Monitor loader and does not appear at the load address.

The boot header has the following format:

```
/*-----+
| Boot header.
+-----*/
typedef struct boot_block {
    unsigned long    magic;
    unsigned long    dest;
    unsigned long    num_512blocks;
    unsigned long    debug_flag;
    unsigned long    entry_point;
    unsigned long    reserved[3];
} boot_block_t;
```

- **magic** identifies this image as a legitimate ROM Monitor image and must have the value X'0052 504F'.
- **dest** is the target address for the image (after the boot header is stripped off).
- **num\_512blocks** - Boot images are padded to a multiple of 512 byte blocks. This field specifies the number of blocks.
- **debug\_flag** controls whether the ROM Monitor debugger gets control before the loaded image starts. If the value is 0x0, the image runs immediately. If 0x01, the debugger gains control as soon as the load is complete.
- **entry\_point** specifies the address where the image will receive control.



## 602 EVB Kit Bill of Materials

---

This appendix presents the bill of materials for the EVB and its components:

### C.1 Bill of Materials

The table headings are described below.

**QTY**

- Quantity Used

**Vendor PN**

- Supplier Code or Part Number From the Vendor Catalog

**Description**

- The Component Description

**Comp Code**

- System Assigned Component Code (R1, C33, U1, etc)

**Vendor**

- Supplier Code of the Part

<b>QTY</b>	<b>Vendor PN</b>	<b>Description/Comp Code</b>	<b>Vendor</b>
1	42H4081	<b>Raw Card</b>	IBM
1	602 IBM RISK Processor	(PPC602-F1-040 PQFP144 40 Mhz) (144 pin PQFP) <b>U33</b>	IBM
3	SN7406D	Open collector hex inverter (14 pin 250 mill SMT plastic) <b>U3 U15 U20</b>	TI/National
8	74ABT16501-DL	18 BIT UNIVERSAL BUS Transceiver W/3ST Outputs (300 mill shrink small outline plastic) <b>U51 U52 U53 U54 U57 U58 U59 U60</b>	TI
1	74ALS04D	Hex inverter (14 pin 250 mill SMT plastic) <b>U74</b>	TI/National

<b>QTY</b>	<b>Vendor PN</b>	<b>Description/Comp Code</b>	<b>Vendor</b>
4	SN74ALS244D	Octal Buffer Line driver/ Line receiver <b>(20 pin 300 mill SMT plastic)</b> <b>U17 U64 U65 U73</b>	TI/National
2	SN74F08D	Quadruple 2 input positive AND gate <b>(14 pin 250 mill SMT plastic)</b> <b>U22 U26</b>	TI/National
2	74F251D	8 input Multiplexer <b>(16 pin 250 mill SMT plastic)</b> <b>U14 U16</b>	Motorola
1	SN74F74D	Dual D type positive edge triggered flip flop <b>(16 pin 250 mill SMT plastic)</b> <b>U72</b>	TI/National
4	74FCT162260-ETPV	FAST CMOS 12 BIT TRI-PORT BUS EXCHANGER <b>(56 pin shrink small outline SMT plastic SO56-1)</b> <b>U36 U37 U40 U41</b>	IDT
6	74FCT162H272-ETPV	FAST CMOS 12 BIT SYNCHRONOUS BUS EXCHANGER <b>(56 pin shrink small outline SMT plastic SO56-1)</b> <b>U24 U25 U30 U31 U32 U34</b>	IDT
2	74LVX3L383WM	10 BIT LOW POWER BUS EXCHANGE SWITCH <b>(24 pin molded small outline 300 mill SMT plastic)</b> <b>U48 U49</b>	National
8	74LVX3L384WM	10 BIT LOW POWER BUS SWITCH <b>(24 pin molded small outline 300 mill SMT plastic)</b> <b>U18 U19 U28 U29 U38 U39 U42 U43</b>	National
4	AM29F040-120JC	512k x 8, 5 volt only sector erase FLASH memory (32 pin PLCC) <b>U67 U68 U69 U70</b>	AMD
4	822273-1	32 Pin PLCC sockets for flash memory	AMP
	A2146-ND		DigiKey

<b>QTY</b>	<b>Vendor PN</b>	<b>Description/Comp Code</b>	<b>Vendor</b>
169	ECU-V1H104KBW	0.1U 10% 50 volts Chip capacitor	Panasonic
	PCC104BCT-ND	(1206 SMT Ceramic Chip capacitor)	DigiKey
		<b>C22 C25 C26 C27 C30 C31 C34 C36 C37 C38 C39 C40 C41 C42 C43 C44 C45 C46 C47 C48 C49 C50 C51 C52 C53 C54 C55 C56 C57 C58 C59 C60 C61 C62 C63 C64 C65 C66 C67 C68 C69 C70 C71 C72 C73 C74 C75 C76 C77 C78 C79 C80 C81 C82 C83 C84 C85 C86 C87 C88 C89 C90 C91 C92 C93 C94 C95 C96 C97 C98 C99 C100 C101 C102 C103 C104 C105 C106 C107 C108 C109 C110 C111 C112 C113 C114 C115 C116 C117 C118 C119 C120 C121 C122 C123 C124 C125 C126 C127 C128 C129 C130 C131 C132 C133 C134 C135 C136 C137 C138 C139 C140 C141 C142 C143 C144 C145 C146 C147 C148 C149 C150 C151 C152 C153 C154 C155 C156 C157 C158 C159 C160 C161 C162 C163 C164 C165 C166 C167 C168 C169 C170 C171 C172 C173 C174 C175 C176 C177 C178 C179 C180 C181 C182 C183 C184 C185 C186 C187 C188 C189 C190 C191 C193 C194 C195 C196 C198 C199</b>	
13	ECS-H1CC106R	10U 20% 16 volts Tantalum chip capacitor	Panasonic
	PCT3106CT-ND	(2312 SMT Chip capacitor) <b>C4 C5 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17</b>	DigiKey
7	PCC101CCT-ND	22U 20% 16 volts Tantalum chip capacitor	Panasonic
	PCT3226CT-ND	(2816 SMT Chip capacitor) <b>C1 C2 C3 C6 C18 C23 C24</b>	DigiKey
7	PCC101CCT-ND	100P 5% 50 volts	DigiKey
	1206A101JAT1A	(1206 SMT Ceramic chip capacitor) <b>C28 C29 C32 C33 C35 C192 C197</b>	AVX
3	PCC200CCT-ND	20P 5% 50 volts Chip capacitor	DigiKey
	12061A200JATMA	(1206 SMT Ceramic chip capacitor) <b>C19 C20 C21</b>	AVX

<b>QTY</b>	<b>Vendor PN</b>	<b>Description/Comp Code</b>	<b>Vendor</b>
1	89F4286	5 PIN DIN RECPT RT ANGLE Connector	Newark
	211450-1	(Power input) <b>J3</b>	AMP
1	SE2509CT-ND	20 MHZ Crystal for Ethernet	DigiKey
	MA-505-20.00M-C2	(MA-505 package) <b>Y2</b>	EPSON
1	CY7B991-5JC	PROGRAMMABLE SKEW CLOCK BUFFER (PLCC32) <b>U10</b>	CYPRESS
1	745984-3	15 PIN D-SHELL S&L TAIL Connector - AUI <b>J15</b>	AMP
2	A2100-ND	9 PIN D-SHELL S&L TAIL Connector - Serial Port <b>J25 J27</b>	DigiKey
	747840-4		AMP
1	DMC16117A	DOT MATRIX LCD 16x1 character DISPLAY <b>DS1</b>	OPTREX
	OP116NB-ND		DigiKey
1	DS1233-Z10	5 volt EconoReset (SOT223 package) <b>U23</b>	DALLAS
4	822019-4	72 Pin Simm socket <b>U44 U45 U46 U47</b>	AMP
1	154 004	4 AMP Fuse for 5 volt supply (Omni-Blok Nano surface mount fuse) <b>F2</b>	Lit-tlefuse
1	154 001	1 AMP Fuse for 12 volt supply (Omni-Blok Nano surface mount fuse) <b>F1</b>	Lit-tlefuse

<b>QTY</b>	<b>Vendor PN</b>	<b>Description/Comp Code</b>	<b>Vendor</b>
9	2440-6112TG	2 pin header male <b>J4 J5 J6 J10 J16 J17 J18 J19 J20</b>	3M
9	<b>929950-00-I</b>	<b>2 pin program jumper</b> <b>J4 J5 J6 J10 J16 J17 J18 J19 J20</b>	<b>3M</b>
1	PS-16-DS-WXA	RISCWATCH 2x8 keyed header (pre-keyed) (not-keyed) <b>J13</b>	Thomas & Betts
	S2011-16-ND		Digi-Key
12	2440-6112TG	1 pin header male <b>J2 J7 J8 J9 J11 J12 J14 J21 J22 J24 J26 J28</b>	3M
2	LN342GP )	LED indicator (green (2 pin - pin in hole pkg) <b>D3 D5</b>	Panasonic
	HLMP-S500		HP
1	LN442YP	LED indicator (yellow) (2 pin - pin in hole pkg) <b>D4</b>	Panasonic
	HLMP-S300		HP
2	LN242RP	LED indicator (red) (2 pin - pin in hole pkg) <b>D1 D2</b>	Panasonic
	HLMP-S100		HP
1	LT1117-3.3	3.3 volt 800ma low dropout positive regulator (SOT223 package) <b>U27</b>	LINEAR
2	MAX208CWG	QUAD RS232 TRANSCEIVER (24 pin 300 mill wide SO package) <b>U61 U71</b>	Maxim
	ADM208AR		Analog Devices

QTY	Vendor PN	Description/Comp Code	Vendor
1	MB86964PFV-G	ETHERNET CONTROLLER (100 pin shrink quad flat pack FPT-100P-M05) <b>U62</b>	Fujitsu
2	MBRD640CT	Schottky Rectifier 6 amps 20 volts (case 369a-13 DPAK) <b>Q1 Q2</b>	Motorola
1	MCR706A	4 amp, 400 volt SCR (case 369a STYLE 5 DPAK) <b>CR2</b>	Motorola
1	MK48T02-120PC1	2KX8 TIMEKEEPER SRAM (24 pin 600 mill DIP) <b>U35</b>	SGS Thomson
1	91895-103	MICROPAX 200 Connector (200 pin stradle connector with solder bumps) <b>J1</b>	Berg
1	MX045HS-7.3728	7.3728MHZ Oscillator (4 pin half dip package) <b>Y3</b>	CTS
	OECS-73-2-A401A		DigiKey
1	MX045HS-40	40MHZ Oscillator (4 pin half dip package) <b>Y1</b>	CTS
2	545568-4	Crystal oscillator sockets (4 pin half dip package) <b>Y1 Y3</b>	AMP
1	PC16552DV	Dual Universal Asynchronous receiver/transmitter with FIFO (44 pin PLCC) <b>U66</b>	National
1	PE65723	ISOLATION TRANSFORMER FOR AUI (SMT package - type Y mechanical) <b>T1</b>	Pulse Engineering
1	PE65745	PULSE TRANSFORMER FOR 10 BASE - T (SMT package - type P mechanical) <b>U55</b>	Pulse Engineering

QTY	Vendor PN	Description/Comp Code	Vendor
2	A2216	SPDT SWITCH Pushbutton (sealed) (TP32P003000 with white dot on side) (3 Pin - pin in hole) <b>U2 U12</b>	APEM
		(TP32P003000 0 with white dot on side)	APEM
1	23AR5K	5k 1 turn adjustable Pot (3 Pin SMT hook lead) <b>R34</b>	Beckman Industrial
	3314J-502E CT-ND		DigiKey
6	CRCW1206- 100-137	137 ohms 1% .125W (SMT 1206) <b>R64 R74 R112 R159 R180 R21</b>	DALE
2	CRCW1206- 100-37	37.4 ohms 1% .125W (SMT 1206) <b>R36 R149</b>	DALE
2	CRCW1206- 100-75	75 ohms 1% .125W (SMT 1206) <b>R140 R160</b>	DALE
1	CRCW1206- 100-1242FT	12.4K ohms 1% .125W (SMT 1206) <b>R176</b>	DALE
1	CRCW1206- 100-10	10 ohms 1% .125W (SMT 1206) <b>R33</b>	DALE
2	CRCW1206- 100-0	0.0 ohms 1% .125W (SMT 1206) <b>R50 R53</b>	DALE
22	CRCW1206- 100-4991FT	4.99K ohms 1% .125W (SMT 1206) <b>R3 R4 R5 R7 R8 R9 R10 R11 R12 R13 R14 R15 R16 R20 R52 R55 R57 R59 R61 R62 R78 R191</b>	DALE

QTY	Vendor PN	Description/Comp Code	Vendor
122	CRCW1206-100-1002FT	10K ohms 1% .125W (SMT 1206) <b>R17 R18 R19 R23 R24 R25 R26 R27 R28 R29 R30 R32 R35 R40 R43 R45 R46 R47 R48 R66 R67 R68 R69 R70 R72 R73 R75 R77 R79 R80 R81 R82 R83 R84 R85 R86 R88 R89 R90 R91 R92 R93 R94 R95 R96 R97 R98 R99 R100 R102 R106 R110 R114 R115 R116 R118 R119 R120 R121 R122 R123 R124 R130 R131 R132 R133 R134 R136 R141 R142 R143 R144 R145 R146 R147 R148 R150 R151 R152 R153 R154 R155 R156 R157 R158 R161 R162 R166 R167 R168 R169 R170 R171 R172 R173 R174 R177 R179 R181 R182 R183 R184 R185 R186 R187 R188 R189 R190 R192 R193 R194 R195 R196 R197 R198 R199 R200 R201 R202 R203 R204 R208</b>	DALE
5	CRCW1206-100-191	191 ohms 1% .125W (SMT 1206) <b>R65 R71 R111 R164 R178</b>	DALE
2	CRCW1206-100-80	80.6 ohms 1% .125W (SMT 1206) <b>R206 R207</b>	DALE
23	CRCW1206-100-1001FT	1K ohms 1% .125W (SMT 1206) <b>R22 R31 R37 R38 R39 R41 R42 R44 R76 R87 R101 R103 R107R108 R113 R125 R126 R127 R128 R129 R137 R138 R205</b>	DALE
3	CRCW1206-100-78	78.7 ohms 1% .125W (SMT 1206) <b>R135 R139 R175</b>	DALE
2	CRCW1206-100-49R9FT	49.9 ohms 1% .125W (SMT 1206) <b>R163 R165</b>	DALE
5	CRCW1206-100-4750FT	470 ohms 1% .125W (SMT 1206) <b>R2 R6 R49 R51 R63</b>	DALE
1	555164-1	RJ45 Connector <b>J23</b>	AMP

<b>QTY</b>	<b>Vendor PN</b>	<b>Description/Comp Code</b>	<b>Vendor</b>
1	TC55257CFL	32KX8 -70 ns STATIC RAM <b>(SOP28-P-450) U63</b>	Toshiba
	MS62256L-70FC		MOSEL-VITELIC
1	AT17C128	FPGA Configuration EE Prom (20 Pin PLCC) <b>U50</b>	Atmel
	XC17128D-PC20C	OTP version	Xilinx
1	822269-1	PLCC-20 socket for Xilinx Prom <b>U50</b>	AMP
	A2140-ND		DigiKey
2	XC3190A-2PQ160C	Xilinx LOGIC CELL ARRAY <b>U21 U56</b>	Xilinx
1	DL5232BCT-ND	5.6 volt ZENER diode (surface mount DL35 package) <b>CR1</b>	Digi-Key
1	SW201KA0012 F01	5 volt / 12 volt power supply ( not board mounted )	AULT
	87G4509		IBM
1	BBL-114-G-E	14 pin bi-di header strip ( display connector )	
1	SS-114-S-2	14 pin header ( display connector )	
<b>Memory</b>			

<b>QTY</b>	<b>Vendor PN</b>	<b>Description/Comp Code</b>	<b>Vendor</b>
<b>1 to 4</b>	15601	<b>1Mx32 EDO 60ns access</b>	<b>VISIONTECH</b>
	IBM11D1325LC -60	1Mx32 EDO 60ns access	IBM
<b>1 to 4</b>	IBM11D2325LC -60	2Mx32 EDO 60ns access	IBM
<b>1 to 4</b>	15631	4Mx32 EDO 60ns access	VISION- TECH
	IBM11D4325BB -60	4Mx32 EDO 60ns access	IBM
<b>1 to 4</b>	IBM11D8325BB -60	8Mx32 EDO 60ns access	IBM
<b>Hardware</b>			
2	207952-3	Jackscrew	AMP
5	N3906-632-0	Nylon Standoffs (6-32x .5)	Globe Elec- tronic Hard- ware
5	Nylon Screws	Nylon Screws (6-32x .5)	
(Note: Does not include the raw card or power supply.)			

## A

Alignment Exception Support Library 9-1  
ANSI C I/O Library 9-1  
ANSI C Library 9-1  
ANSI C Math Library 9-1  
async safe 10-1  
async\_init() function 10-9  
asyncLib.a library 9-4

## B

biosenet\_attach() function 10-10  
Block Buffer Library 9-1  
book  
    conventions used xx  
    highlighting xxi  
    numeric xxi  
    syntax diagrams xxi  
    how organized xx  
    who should use this book xix  
Boot Library 9-1, 9-4  
booting the processor 5-22

## C

C++ runtime support library 9-2  
cancel safe 10-2  
Clock Support Library 9-2  
clock\_set() function 10-12  
clockchip\_nvram\_read() function 10-15  
clockchip\_nvram\_write() function 10-16  
clockchip\_set() function 10-13, 10-14, 10-17  
clockchip\_set\_calibration() function 10-18  
clockchip\_start() function 10-19  
clockchip\_stop() function 10-20  
clockLib.a library 9-5  
clockLib\_init() function 10-21  
connecting the EVB hardware 5-15  
connectors  
    ethernet 5-3  
    expansion interface 5-7  
    power 5-18  
RISCTrace 5-6  
RISCWatch 400 JTAG 5-6  
serial port 5-3

conventions used xx  
    highlighting xxi  
    numeric xxi  
    syntax diagrams xxi

## D

dbg\_ioLib\_init() function 10-22  
dcache\_flush() function 10-23  
dcache\_invalidate() function 10-24  
Debug Support Library 9-2  
Device and File Support Library 9-2  
device drivers  
    asynchronous 9-6  
    Ethernet 9-12  
DOS File System Support Library 9-2  
driver\_install  
    async\_init 9-6  
    enet\_init 9-12  
Dynamic Loader Library 9-2

## E

enet\_init function 10-25  
enetLib.a 9-5  
Ethernet 9-12  
Ethernet Device Driver Installation 9-12  
ext\_int\_disable() function 10-26  
ext\_int\_enable() function 10-27  
ext\_int\_install() function 10-28  
ext\_int\_query() function 10-29

## F

Federal Communications Commission (FCC)  
    Statement xxiii  
File Transfer Protocol Support Library 9-2  
Flash update utility 7-31  
Floating Point Emulation Library 9-2  
functions  
    async\_init() 10-9  
    biosenet\_attach() 10-10  
    clock\_set() 10-12  
    clockchip\_get() 10-13  
    clockchip\_get\_calibration() 10-14

clockchip\_nvram\_read() 10-15  
clockchip\_nvram\_write() 10-16  
clockchip\_set() 10-13, 10-14, 10-17, 10-  
21  
clockchip\_set\_calibration() 10-18  
clockchip\_start() 10-19  
clockchip\_stop() 10-20  
clockLib\_init() 10-21  
dbg\_ioLib\_init() 10-22  
dcache\_flush() 10-23  
dcache\_invalidate() 10-24  
enet\_init 10-25  
ext\_int\_disable() 10-26  
ext\_int\_enable() 10-27  
ext\_int\_install() 10-28  
ext\_int\_query() 10-29  
ioLib\_init() 10-30  
lcd\_init() 10-31  
memcpy\_io() 10-32  
ppcAbend() 10-33  
ppcAndMsr() 10-34  
ppcCntrlzw() 10-35  
ppcDcbf() 10-36  
ppcDcbi() 10-37  
ppcDcbst() 10-38  
ppcDcbz() 10-39  
ppcEieio() 10-40  
ppcHalt() 10-41  
ppclcbi() 10-42  
ppclsync() 10-43  
ppcMdar() 10-44  
ppcMfdbat0l() - ppcMfdbat3l() 10-45  
ppcMfdbat0u() - ppcMfdbat3u() 10-46  
ppcMfdec() 10-47  
ppcMfdsisr() 10-48  
ppcMfear() 10-49  
ppcMfesasrr() 10-50  
ppcMfgpr1() 10-51  
ppcMfgpr2() 10-52  
ppcMfhod0() 10-53, 10-54  
ppcMfiabr() 10-55  
ppcMfibat0l() - ppcMfibat3l() 10-57  
ppcMfibat0u() - ppcMfibat3u() 10-58  
ppcMfibr() 10-56  
ppcMflt() 10-59  
ppcMfmsr() 10-60

ppcMfpvr() 10-61  
ppcMfsdr1() 10-62  
ppcMfsebr() 10-63  
ppcMfser() 10-64  
ppcMfsp() 10-65  
ppcMfsprg1() • ppcMfsprg3() 10-66  
ppcMfsprg1() • ppcMtsprg3() 10-90  
ppcMfsr() 10-67  
ppcMfsrr0() 10-68  
ppcMfsrr1() 10-69  
ppcMftb() 10-70  
ppcMftcr() 10-71  
ppcMtدار() 10-72  
ppcMtدار0l() - ppcMtدار3l() 10-73  
ppcMtدار0u() - ppcMtدار3u() 10-74  
ppcMtdec() 10-75  
ppcMtdeisr() 10-76  
ppcMtear() 10-77  
ppcMtexasrr() 10-78  
ppcMtihod0() 10-79  
ppcMtibr() 10-80  
ppcMtibat0l() - ppcMtibat3l() 10-82  
ppcMtibat0u() - ppcMtibat3u() 10-83  
ppcMtibr() 10-81  
ppcMtlt() 10-84  
ppcMtmsr() 10-85  
ppcMtsdr1() 10-86  
ppcMtsebr() 10-87  
ppcMtser() 10-88  
ppcMtsp() 10-89  
ppcMtssr() 10-91  
ppcMtssrr1() 10-93  
ppcMttsb() 10-94  
ppcMttscr() 10-95  
ppcOrMsr() 10-96  
ppcSync() 10-97  
s1dbprintf() 10-98  
s2dbprintf() 10-99  
timertick\_install() 10-100  
timertick\_remove() 10-101  
vs1dbprintf() 10-102

## H

### hardware

evaluation board bill of materials C-1  
hardware components 1-1

cables and power supply 1-1  
evaluation board 1-1  
host system requirements  
PC 2-2  
RS/6000 2-1  
Sun 2-3

## I

I/O control 9-9, 9-18  
IBM Embedded Systems Solution Center xxii  
initialization 9-19  
    board bootstrap 9-20  
Input/output Support Library 9-2  
installing  
ioLib.a library 9-4  
ioLib\_init() function 10-30

## J

jumpers  
    setting 5-13

## K

Kernel Abstract Data Types Library 9-2

## L

LCD Library 9-2  
lcd\_init() function 10-31  
library description  
    asyncLib.a 9-4  
    clockLib.a 9-5  
    enetLib.a 9-5  
    ioLib.a 9-4  
    ppcLib.a 9-5  
    rtx.o 9-3  
    rtxLib.a 9-3  
    tickLib.a 9-5

## M

memcpy\_io() function 10-32

## N

Network Support Library 9-2  
NFS Support Library 9-2

## O

Opening and Closing Ethernet Files 9-12

opening asynchronous communication ports 9-8  
OpenShell 9-2  
OS Open kernel extensions 9-3  
OS Open minimal kernel 9-3

## P

PC host configuration 4-7  
ethernet setup 4-14  
ethernet setup for Windows 3.1 4-10  
ethernet setup for Windows 95 4-12  
ethernet setup for Windows NT 3.51 4-13  
serial port setup 4-8  
PC installation 3-4  
    RISCWatch debugger 3-6  
    software support package 3-4  
PCMCIA ATA/IDE 9-2  
PCMCIA card services/enabler 9-2  
PCMCIA socket sevices 9-3  
polled asynchronous I/O 9-11  
PowerPC Low Level Access Support Library 9-2  
PowerPC Low-Level Processor Access Support Library 9-5  
ppcAbend() function 10-33  
ppcAndMsr() function 10-34  
ppcCntlzw() function 10-35  
ppcDcbf() function 10-36  
ppcDcbi() function 10-37  
ppcDcbst() function 10-38  
ppcDcbz() function 10-39  
ppcEieio() function 10-40  
ppcHalt() function 10-41  
ppcIcbc() function 10-42  
ppcLib.a library 9-5  
ppclsync() function 10-43  
ppcMdadar() function 10-44  
ppcMfdbat0l() - ppcMfdbat3l() function 10-45  
ppcMfdbat0u() - ppcMfdbat3u() function 10-46  
ppcMfdec() function 10-47  
ppcMfdsisr() function 10-48  
ppcMfear() function 10-49, 10-50  
ppcMfgpr1() function 10-51  
ppcMfgpr2() function 10-52

ppcMfhid0() function 10-53  
ppcMfhid1() function 10-54  
ppcMfiabr() function 10-55  
ppcMfibat0l() - ppcMfibat3l() function 10-57  
ppcMfibat0u() - ppcMfibat3u() function 10-58  
ppcMfibr() function 10-56  
ppcMflt() function 10-59  
ppcMfmsr() function 10-60  
ppcMfpvr() function 10-61  
ppcMfsdr1() function 10-62  
ppcMfsebr() function 10-63  
ppcMfser() function 10-64  
ppcMfsp() function 10-65  
ppcMfsprg0() • ppcMfsprg3() function 10-66  
ppcMfsr() function 10-67  
ppcMfsrr0() function 10-68  
ppcMfsrr1() function 10-69  
ppcMftb() function 10-70  
ppcMftcr() function 10-71  
ppcMtddar() function 10-72  
ppcMtddbat0l() - ppcMtddbat3l() function 10-73  
ppcMtddbat0u() - ppcMtddbat3u() function 10-74  
ppcMtdec() function 10-75  
ppcMtddisr() function 10-76  
ppcMttear() function 10-77  
ppcMttesasrr() function 10-78  
ppcMtthid0() function 10-79  
ppcMtibr() function 10-80  
ppcMtibat0l() - ppcMtibat3l() function 10-82  
ppcMtibat0u() - ppcMtibat3u() function 10-83  
ppcMtibr() function 10-81  
ppcMtlt() function 10-84  
ppcMtmsr() function 10-85  
ppcMttsdr1() function 10-86  
ppcMtsebr() function 10-87  
ppcMtser() function 10-88  
ppcMtsp() function 10-89  
ppcMtssprg0() • ppcMtssprg3() function 10-90  
ppcMttsr() function 10-91  
ppcMttsrr1() function 10-93  
ppcMttsb() function 10-94  
ppcMttscr() function 10-95

ppcOrMsr() function 10-96  
ppcSync() function 10-97  
ptrace  
definitions A-4  
RD\_ATTACH A-5  
RD\_CONTINUE A-6  
RD\_DETACH A-7  
RD\_FILL A-8  
RD\_KILL A-9  
RD\_LDINFO A-10  
RD\_LOAD A-12  
RD\_LOGIN A-13  
RD\_LOGOFF A-14  
RD\_READ\_D A-15  
RD\_READ\_GPR A-16, A-17, A-22  
RD\_READ\_GPR\_MULT A-18  
RD\_READ\_I A-19  
RD\_READ\_I\_MULT A-20  
RD\_READ\_SPR A-21  
RD\_STATUS A-23  
RD\_STOP\_APPL A-24  
RD\_WAIT A-25  
RD\_WRITE\_BLOCK A-26  
RD\_WRITE\_D A-27  
RD\_WRITE\_GPR A-28, A-29, A-32  
RD\_WRITE\_I A-30  
RD\_WRITE\_SPR A-31  
RL\_LDINFO A-33  
RL\_LOAD\_REQ A-34  
MSGDATA structure A-1  
overview A-1

## Q

Queue Library 9-2

## R

RAM Disk Library 9-2  
Rate Monotonic Scheduling (RMS) Library 9-2

Real\_time Executive 9-3

Real-time Clock Interface Support Library 9-5

related publications xxiii

Remote Source Level Debug Library 9-2

Ring Buffer Library 9-2

ROM monitor

accessing 7-8  
 bootp and tftp configuration for loads 7-2  
     PC 7-4, 7-6  
     RS/6000 7-2  
 communication features 7-2  
 menus 7-10  
     changing IP addresses 7-16  
     disabling the automatic display 7-22  
     displaying the current configuration 7-23  
     entering the debugger 7-20  
     exiting the main menu 7-29  
     initial ROM monitor menu 7-11  
     saving the current configuration 7-24  
     selecting boot devices 7-14  
     selecting power-on tests 7-12  
     using the ping test 7-18  
 source code 7-1  
 user functions 7-31  
**ROM Monitor Ethernet Interface Library** 9-1  
**ROM monitor load format**  
**boot header** B-3  
**overview** B-1  
**section types** B-1  
     data section B-3  
     first section B-2  
     symbol section B-3  
**sections types**  
     text section B-3  
**RPC Support Library** 9-2  
**RS/6000 host configuration** 4-1  
     ethernet setup 4-5  
     serial port setup 4-1  
**RS/6000 installation** 3-1  
     RISCWatch debugger 3-4  
     software support package 3-1  
**rtx.o library** 9-3  
**rtxLib.a library** 9-3  
**Runtime Library** 9-2

**S**

**s1dbprintf()** function 10-98  
**s2dbprintf()** function 10-99  
**sample applications**  
     **overview** 8-1  
     **resolving problems** 8-8

bootp and tftp servers 8-8  
     using the ping test 8-8  
**ROM monitor flash image** 8-2  
     **using** 8-4  
         Dhrystone benchmark 8-5  
         timesamp program 8-7  
         usr\_samp program 8-6  
**SCSI Support Library** 9-2  
**Serial Port Support Library** 9-4  
**Serial Support Library** 9-2  
**software support package** 1-2  
     application libraries and tools 1-3  
     Dhrystone benchmark 1-3  
     High C/C++ compiler 1-3  
     RISCWatch 400 debugger 1-2  
     ROM monitor 1-2  
**Software Timer Tick Support Library** 9-5  
**Sun host configuration** 4-13  
**Sun installation**  
     RISCWatch debugger 3-9  
     software support package 3-6  
**Symbol Support Library** 9-3

**T**

**TCP/IP Protocol Support Library** 9-3  
**Telnet Client Support Library** 9-3  
**Telnet Daemon Support Library** 9-3  
**terminal emulator** 5-19  
     PC terminal emulation 5-20  
     Windows 3.1 & NT 5-20  
     Windows 95 5-20  
**RS/6000 terminal emulation** 5-19  
     Sun terminal emulation 5-21  
**tickLib.a library** 9-5  
**Timer Tick Support** 9-3  
**timertick\_install()** function 10-100  
**timertick\_remove()** function 10-101  
**tools** 9-20  
     eimgbld 9-25, 9-26  
     elf2rom 9-20  
     hbranch 9-23  
**Trivial File Transfer Protocol Library** 9-3  
**TTY Support Library** 9-3

**V**

**vs1dbprintf()** function 10-102

**W**

writing calls on asynchronous ports 9-8, 9-

18



